Unit Edge-Length Rectilinear Drawings with Crossings and Rectangular Faces

Patrizio Angelini
Department of Mathematics, Natural, and Applied Sciences, John Cabot University, Italy

Carla Binucci
Department of Engineering, University of Perugia, Italy

Giuseppe Di Battista
Department of Engineering, University of Roma Tre, Italy

Emilio Di Giacomo
Department of Engineering, University of Perugia, Italy

Walter Didimo
Department of Engineering, University of Perugia, Italy

Fabrizio Grosso
CeDiPa, University of Perugia, Italy

Giacomo Ortali
Department of Engineering, University of Perugia, Italy

Ioannis G. Tollis
Department of Computer Science, University of Crete, Greece

Abstract

Unit edge-length drawings, rectilinear drawings (where each edge is either a horizontal or a vertical segment), and rectangular face drawings are among the most studied subjects in Graph Drawing. However, most of the literature on these topics refers to planar graphs and planar drawings. In this paper we study drawings with all the above nice properties but that allow edge crossings; we call them Unit Edge-length Rectilinear drawings with Rectangular Faces (UER-RF drawings). We consider crossings as dummy vertices and apply the unit edge-length convention to the edge segments connecting any two (real or dummy) vertices. Note that UER-RF drawings are grid drawings (vertices are placed at distinct integer coordinates), which is another classical requirement of graph visualizations. We present several efficient and easily implementable algorithms for recognizing graphs that admit UER-RF drawings and for constructing such drawings if they exist. We consider restrictions on the degree of the vertices or on the size of the faces. For each type of restriction, we consider both the general unconstrained setting and a setting in which either the external boundary of the drawing is fixed or the rotation system of the graph is fixed as part of the input.

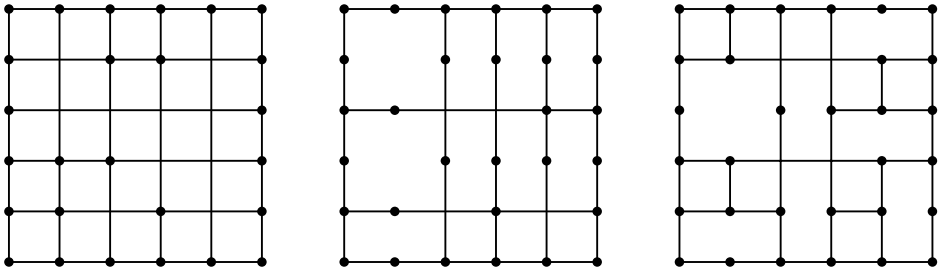
Keywords and phrases Rectilinear drawings, Unit-length drawings, Bounded-degree graphs

Digital Object Identifier 10.57717/cgt.v5i3.97

Related Version A preliminary version of this work appeared at European Workshop on Computational Geometry (EuroCG'25).

Funding Research partially supported by MUR of Italy (PRIN Project no. 2022ME9Z78 – NextGRAAL and PRIN Project no. 2022TS4Y3N – EXPAND). *Fabrizio Grosso*: Supported by Ce.Di.Pa. - PNC Programma unitario di interventi per le aree del terremoto del 2009-2016 - Linea di intervento 1 sub-misura B4 - "Centri di ricerca per l'innovazione" CUP J37G22000140001.

Acknowledgements This work started at the Bertinoro Workshop on Graph Drawing BWGD 2024.



(a) 5×5 **(b)** 5×5 **(c)** 5×5

Figure 1 UER-RF drawings: with Unit Square Faces (a), with no internal degree-3 vertex (b), and in the general setting (c). All drawings occupy a 5×5 grid area.

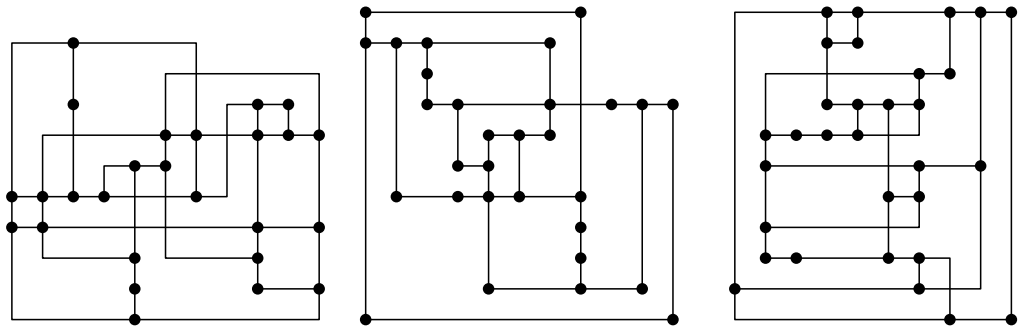
1 Introduction

In an *orthogonal drawing* of a graph all vertices are points of the plane (usually at integer grid coordinates) and edges are chains of horizontal and vertical segments. An orthogonal drawing where each edge is drawn as a single segment (either horizontal or vertical) is also called a *rectilinear drawing* (see, e.g.,[15, 16]). Planar rectilinear drawings where all edges have unit length and all faces are convex (i.e., rectangles), adhere to several of the most celebrated Graph Drawing aesthetics. Further, such drawings can be translated in the plane so that their vertices have integer coordinates. Motivated by these considerations, a recent paper investigated these types of planar layouts [2, 3].

We extend the above drawing convention to non-planar drawings, considering each crossing as a "dummy" vertex and applying the unit edge-length constraint to the edge segments connecting any two vertices, either real or dummy. In this setting, we study *Unit Edge-length Rectilinear drawings with Rectangular Faces* (*UER-RF drawings*), i.e., rectilinear drawings in which all edge segments have unit length and all faces (including the external one) are rectangles. Examples of UER-RF drawings are presented in Figure 1. Note that, in addition to guaranteeing edges without bends and rectangular faces, UER-RF drawings (when they exist) are typically more compact than orthogonal grid drawings computed with classical algorithms, which aim to minimize crossings and bends at the same time; for a visual comparison, see for example Figure 2.

The problem we study is related to several challenging problems. Recognizing graphs that admit planar straight-line drawings with all edges of the same length is NP-hard [6, 7, 13] and, even stronger, $\exists\mathbb{R}$-complete [1, 28]. Testing whether a graph is rectilinear planar is NP-hard [15, 16], as well as recognizing graphs that admit unit edge-length drawings with integer coordinates is NP-complete even for trees [5, 17, 18]. On the other hand, recognizing planar graphs that admit rectilinear drawings with rectangular faces (with unconstrained edge lengths) can be done in polynomial time if the planar embedding is part of the input [8, 9, 25, 26, 29]; see also [14, 22, 23]. Additionally, observe that UER-RF drawings are RAC drawings (see, e.g., [4, 11]) and that non-planar drawings are intensively studied in the beyond planarity area (for surveys see [12, 19, 27]).

Alegría et al. [3] show that recognizing planar graphs admitting planar UER-RF drawings is feasible in polynomial time. We aim to extend this result to non-planar graphs. We study the recognition and the construction of UER-RF drawings under various conditions: when the rotation system of the vertices is part of the input, when it is not, and when the external face is part of the input or not. In our problem, an important role is played by the assignment of angles around the vertices. Requiring rectangular faces enforces 180°



(a) 11×10 **(b)** 11×11 **(c)** 10×11

Figure 2 The same graphs depicted in Figure 1 drawn with the topology-shape-metrics algorithm implemented in the GDToolkit library [10]. Below each drawing we report the width and the height in terms of grid units. These dimensions are at least twice those of the drawings in Figure 1; the three drawings have been scaled-down to fit in the same row.

angles at degree-2 vertices, while the difficulty emerges in the presence of degree-3 vertices. In view of this, we present incremental restricted scenarios in which we are able to solve the problem efficiently, and then give a fixed-parameter-tractable (FPT) algorithm for the general case, parameterized by the number of degree-3 vertices. Additionally, we consider UER-RF drawings in which all faces, except the external one, are unit-area squares. We refer to these drawings as *Unit Edge-length Rectilinear drawings with Unit Square Faces* (*UER-USF drawings*); they have the nice feature of fully exploiting the available rows and columns; see Figure 1a for an example.

Our results. Let G be an n-vertex graph. We prove the following results (see also Table 1 for an overview). We present an $O(n)$-time algorithm that tests whether G admits a UER-USF drawing. The algorithm can be adapted to preserve a prescribed external face (Section 3). We devise polynomial-time algorithms for testing the existence of UER-RF drawings of G in restricted scenarios (Section 4). More precisely,

- We first consider drawings without internal degree-3 vertices and define various algorithms depending on the specific constraints on the external face and on the rotation system. The complexity of these algorithms is $O(n)$ if the four corners of the external face are given, or the rotation system is prescribed, or G has no degree-4 vertex and the external cycle is prescribed; it is $O(n^3)$ if the external cycle is prescribed and $O(n^5)$ in the general case.
- We then study drawings of graphs such that removing the external face yields a collection of paths and cycles (*inner-2-graphs*) and we obtain an algorithm whose running time is $O(n^2)$ if the four corners of the external face are given and $O(n^4)$ in the general case.

We finally present an $O(3^k n^{4.5})$-time algorithm that tests whether G admits a general UER-RF drawing, where k is the number of degree-3 vertices of G (Section 5). The running time reduces to $O(3^k n^2)$ if the four corners of the external face are given and $O(3^k n^4)$ if the external cycle is prescribed.

All our algorithms are easily implementable and can preserve a given rotation system for the vertices, if needed. If the recognition is successful, within the same time complexity of the test, they can construct the corresponding drawing.

		Graph restrictions			Drawing restrictions			
Model	**Face shape**	**Inner-2-graph**	**Max degree-3**	**General**	**Corners**	**External cycle**	**Rotation system**	**Running time**
UER-USF	Unit squares			x				$O(n)$
				x		x		$O(n)$
UER-RF	Rectangular			x	x			$O(n)$
				x			x	$O(n)$
			x			x		$O(n)$
				x		x		$O(n^3)$
				x				$O(n^5)$
		x			x			$O(n^2)$
		x						$O(n^4)$
				x	x			$O(3^k n^2)$
				x		x		$O(3^k n^4)$
				x				$O(3^k n^{4.5})$

Table 1 Overview of our results for the UER-USF and UER-RF models under different graph and drawing restrictions.

2 Basic definitions

Let $V(G)$ and $E(G)$ denote the set of vertices and the set of edges of a graph G, respectively. For a vertex $v \in V(G)$, let $E(v)$ denote the set of edges incident to v and let $N(v)$ denote the set of *neighbors of* v, i.e., the set of vertices adjacent to v. The value $|N(v)|$ is the *degree of* v, and is denoted by $\deg_G(v)$. For a positive integer k, a graph G is a *k-graph* if $\deg_G(v) \leq k$ for each $v \in V(G)$. If $\deg_G(v) = 2$ and $N(v) = \{u, w\}$, *smoothing* v consists of removing v (and its incident edges) from $V(G)$ and adding the edge (u, w) to $E(G)$ (i.e., smoothing a vertex is the reverse of subdividing an edge).

Rectilinear drawings. A *rectilinear drawing* Γ of a graph G is an orthogonal drawing without bends[1], i.e., a drawing with the following properties: (i) each vertex $v \in V(G)$ is mapped to a distinct point p_v of an integer grid; (ii) each edge $e = (u, v) \in E(G)$ is mapped to either a horizontal or a vertical segment s_e connecting p_u and p_v; (iii) if $v \in V(G)$ and $e \in E(G)$, s_e does not intersect p_v unless v is an end-vertex of e. We can assume G to be a 4-graph, as otherwise it does not have a rectilinear drawing. We will denote by $x(v)$ and $y(v)$ the x- and y-coordinates of p_v.

A *vertex of* Γ is either a point that corresponds to a vertex of G, in which case it is called a *real-vertex*, or it is a point where two edges of G cross, in which case it is called a *crossing-vertex*. If Γ has no crossing-vertices, it is a *planar rectilinear drawing*. An *edge of* Γ is a portion of an edge of G delimited by two vertices of Γ, which does not contain other vertices of Γ in its interior. An edge of Γ coincides with an edge e of G when e does not cross any other edges of G in Γ. Drawing Γ divides the plane into connected regions, called *faces*. The *boundary* $\mathcal{B}(f)$ of a face f is the circular sequence of vertices (either real-

[1] In the literature about crossing number, the term "rectilinear" is often used to refer to straight-line drawings of graphs [27].

or crossing-vertices) and edges of Γ that delimit f. The unique infinite region is the *external face* of Γ; the other faces are the *internal faces* of Γ. If f is the external face of Γ, we call $\mathcal{B}(f)$ the *external cycle of* Γ.

Rotation systems. A *rotation system* $\mathcal{R}(G)$ *of* G specifies the clockwise order of the edges in $E(v)$, for each vertex $v \in V(G)$. $\mathcal{R}(v)$ denotes the restriction of $\mathcal{R}(G)$ to v. A (rectilinear) drawing Γ of G determines a rotation system for G; in addition, Γ determines the clockwise order of the edges incident to each crossing-vertex. For a given rotation system $\mathcal{R}(G)$, we say that Γ *preserves* $\mathcal{R}(G)$ if the rotation system determined by Γ coincides with $\mathcal{R}(G)$.

Our models. We study *Unit Edge-length Rectilinear drawings with Rectangular Faces* (*UER-RF drawings*), i.e., rectilinear drawings where all edges have unit length and all faces (including the external one) are rectangles. Note that, in a UER-RF drawing, the external cycle only contains real-vertices. Within the UER-RF model we also define a more restricted model, *Unit Edge-length Rectilinear drawings with Unit Square Faces* (*UER-USF drawings*), where each internal face is a square of unit side; see Figure 1a. The next simple property holds.

▶ **Property 1.** *Let Γ be a UER-RF drawing of a graph G, let C be the external cycle of Γ, and let c_1, c_2, c_3, c_4 be the vertices of C at the corners of Γ. Then: (i) c_1, c_2, c_3, c_4 have degree 2 in G; (ii) each other vertex of C has degree at most 3 in G; (iii) the path in C between two consecutive corners has the same length as the path in C between the other two corners.*

We will assume that the input graph G is biconnected and it is not a cycle, as biconnectivity is necessary for the existence of UER-RF drawings, whereas if G is a cycle the test is trivial: an UER-RF drawing exists if and only if G has even length, and in particular a UER-USF drawing exists if and only if G is a 4-cycle.

3 Unit square faces

In this section we focus on UER-USF drawings. In addition to Property 1, we give two other properties that hold for this model; see Figure 3a. The first one comes from the fact that unit-square faces cannot contain an angle larger than 90°.

▶ **Property 2.** *Let Γ be a UER-USF drawing and let v be a real-vertex of Γ. If v is not on the external cycle of Γ, then $\deg_\Gamma(v) = 4$. Otherwise, v is either a corner of Γ, in which case $\deg_\Gamma(v) = 2$, or $\deg_\Gamma(v) = 3$.*

By Property 2, the external cycle C of any UER-USF drawing has all vertices of degree 3, except for exactly four (the four corners).

▶ **Property 3.** *A UER-USF drawing contains disjoint vertical (horizontal) paths, connecting each non-corner vertex of the top (left) side with a non-corner vertex of the bottom (right) side, possibly traversing internal (degree-4) real-vertices.*

We exploit Properties 1–3 to devise a linear-time recognition and construction algorithm for UER-USF drawings. The algorithm can be adapted to the cases in which the external cycle of the drawing and/or the rotation scheme of the graph are given. We prove the following.

▶ **Theorem 1.** *Let G be an n-vertex 4-graph. There exists an $O(n)$-time algorithm that tests whether G admits a UER-USF drawing. If such a drawing exists, the algorithm constructs one. Moreover, the algorithm can be adapted to preserve a given rotation system and/or to have a prescribed external cycle.*

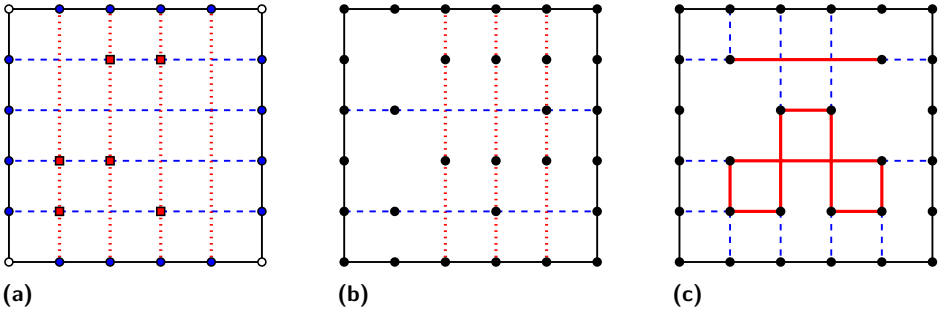


Figure 3 (a) A UER-USF drawing; internal vertices are red squares and external non-corner vertices are blue circles. (b) A UER-RF drawing without internal degree-3 vertices. (c) A UER-RF drawing such that removing the external vertices yields a collection of paths and cycles (red and bold).

To prove Theorem 1 we describe an algorithm that attempts to construct a UER-USF drawing Γ of G in two steps: First, it detects the external cycle C and draws a rectangle Γ_C representing C (Section 3.1); then, it places the remaining vertices of G in the interior of Γ_C (Section 3.2). If any of these two steps fails, the algorithm rejects the instance.

3.1 Choosing and drawing the external face

By Properties 1 and 2, the external cycle C must be composed only of degree-3 vertices, except for exactly four degree-2 vertices; also, the vertices in $V(G) \setminus V(C)$ must have degree 4. To find C, we first check whether G has exactly four degree-2 vertices, which we call *corners*, and then remove all the degree-4 vertices from G; we denote by G' the resulting graph.

Since C must be a spanning subgraph of G', the graph G' must be biconnected; if this is not the case, we can reject the instance. Note that G' consists of the four corners, some other degree-2 vertices (whose third neighbor in $G \setminus G'$ has degree 4 in G), and some degree-3 vertices. For the degree-2 vertices of G', both their incident edges must be part of C. For each degree-3 vertex v of G', we must decide which pair of edges incident to v will belong to C and which one will be a chord of C. If G' has no degree-3 vertex, we can set $C = G'$.

If G' has some degree-3 vertices, we smooth the degree-2 vertices of G' that are not corners; the resulting graph G'' is still biconnected and has all degree-3 vertices except the four corners. We then decide the order in which the corners appear along C. Note that, if two corners are adjacent in G'', they must be consecutive along C. We guess each of the possible circular orders that is consistent with the possible adjacencies. There are at most three circular orders to consider, as we do not need to distinguish between a clockwise and an equivalent counterclockwise order.

Let c_1, c_2, c_3, c_4 be the four corners as they appear in one of the guessed orders. We add to G'' the *corner edges* (c_1, c_2), (c_2, c_3), (c_3, c_4), and (c_4, c_1), if not already present. We claim that, for this order to be compatible with a UER-USF drawing, G'' must have a special structure, which we will exploit to find C. Namely, let C^* be any external cycle of a UER-USF drawing of G in which the four corners are c_1, c_2, c_3, c_4 in this circular order. Let P_i be the path along C^* between c_i and c_{i+1}, for $i \in \{1, 2, 3\}$, and let P_4 the one between c_4 and c_1. By Property 3, all chords of C^* connect either a vertex of P_1 to a vertex of P_3, or a vertex of P_2 to a vertex of P_4 (see Figure 4a). This implies that G'' must be planar and triconnected, so if this is not the case we reject the current guess. Otherwise, G'' admits a unique planar rotation system, up to a flip. In a planar embedding preserving this rotation

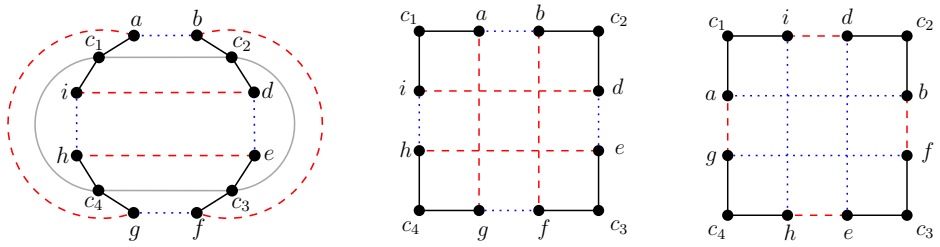


Figure 4 A graph admitting two possible UER-USF drawings, with the same rotation system but different external boundary. Corner edges are grey.

system, the cycle C to be selected as external cycle must be a concatenation of four paths, each sharing a face with a corner edge. This reduces the number of possible cycles to test to at most 16 (two candidate faces for each corner edge). However, each of the two faces incident to a corner edge shares an edge with a face incident to another corner edge. Thus, we must test only two disjoint cycles. Figure 4 shows an example in which both such cycles are valid; the two drawings have the same rotation system.

To conclude, for each of the three possible guessed orders of corners we have obtained at most two candidate cycles of G. Let C be one of such cycles. We check whether C satisfies Condition (iii) of Property 1 in linear time by traversing C; if not, we remove C from the list of candidates. Otherwise, we draw C as a rectangle with the four degree-2 vertices as corners and with all edges having unit length. Then, for each of the constantly many rectangles obtained in this step, we run the algorithm described in Section 3.2.

3.2 Drawing the internal vertices

Let Γ_C be a rectangle representing a cycle C obtained from the algorithm in Section 3.1. To place the vertices of $G \setminus C$ in the interior of Γ_C, we aim to construct the disjoint vertical/horizontal paths between pairs of opposite vertices of C dictated by Property 3. To this aim, we consider the vertices on the top side of Γ_C in left-to-right order. Let u be one of them; we assume as an invariant that all the vertical paths with x-coordinate smaller than $x(u)$ have already been drawn. Moreover, for each vertex v already placed to the left of u, we assume that we have assigned a neighbor of v to each direction (top/bottom/left/right) including its right and bottom neighbors, even if they have not been placed yet. When we assign a vertex v to be the top/bottom/left/right neighbor of another vertex w, we implicitly assume that w is assigned as the bottom/top/right/left neighbor of v. As such, some of the vertices that have not been placed yet may have their left and/or top neighbor already assigned. We proceed as follows.

First step. We consider the vertices of C that lie along the left side of Γ_C; let v be one of them. Clearly, v will not have any left neighbor and its top/bottom neighbors must be its immediate predecessor/successor along C (walking on C counterclockwise), so we can assign its unique remaining neighbor w as its right neighbor. If w belongs to C and is not the unique vertex on the right side of Γ_C with $y(w) = y(v)$, we reject Γ_C. Otherwise, the assignment satisfies the invariant.

General step. We consider the leftmost vertex u on the top side of Γ_C that has not been considered yet. It has no top neighbor, whereas its left and right neighbors must be its immediate predecessor/successor along C. So, we have to assign its other neighbor w as its

bottom neighbor. If the top neighbor of w had already been assigned, or if w belongs to C and is not the unique vertex on the bottom side of Γ_C with $x(w) = x(u)$, we reject Γ_C. Otherwise, we proceed as follows.

If $w \in C$, then it lies on the bottom side of Γ_C; thus, it has no bottom neighbor and its left/right neighbors are determined by C, so the invariant is satisfied. Having completed the vertical path starting at u, the algorithm restarts from the next vertex on the top side.

If $w \notin C$, we describe how to check and assign the neighbors of w distinct from u. By the invariant, the left neighbor of w has been already decided in a previous step. If this is not the case, we can reject Γ_C, otherwise we can place w at x-coordinate $x(u)$ and y-coordinate equal to its left neighbor. It remains to assign the other two neighbors of w, call them w_1 and w_2, as bottom and right neighbors of w. If one of them has already been placed, then it must be either the vertex of C on the bottom side of Γ_C with x-coordinate $x(w)$, or the vertex of C on the right side of Γ_C with y-coordinate $y(w)$, as otherwise we can reject Γ_C. In any positive case, we can uniquely identify the bottom and right neighbors of w. Assume instead that neither w_1 nor w_2 has already been placed. The key observation is that the vertex to be chosen as the bottom neighbor of w must have its left neighbor already assigned, by the invariant. On the contrary, the vertex to be chosen as right neighbor should not have its left neighbor assigned, as we need to assign w to this role. We check whether exactly one of w_1 and w_2 has its left neighbor assigned, and reject Γ_C otherwise. If so, we can uniquely assign the bottom and right neighbors of w. Hence, the invariant holds in all cases. The algorithm continues with the one, w_1 or w_2, that has been selected as the bottom neighbor of w, which is processed in the same way as w.

Proof of Theorem 1. If all vertices on the top side of Γ_C have been processed without rejecting Γ_C, we have a UER-USF drawing Γ of G, as each step of the algorithm fulfills the conditions of Properties 1–3. All rows and columns in Γ are covered by a path connecting two external degree-3 vertices, hence every face of Γ is a unit square. All operations described in Section 3.1 can be performed in linear time, including the tests for planarity [21, 24], biconnectivity, and triconnectivity [20]. Processing a vertex in the algorithm of Section 3.2 requires a constant number of operations on it and on its (at most four) neighbors. Thus, the algorithm runs in linear time. Finally, if the external cycle is prescribed, we check that it satisfies the required conditions, skipping the steps in Section 3.1. If the rotation system is given, we check that it is consistent with the unique rotation system of G'' in the algorithm of Section 3.1, and with the choices performed by the algorithm in Section 3.2, which are unique in all cases.

4 Rectangular faces: restricted scenarios

In this section we deal with UER-RF drawings and consider two different restricted scenarios for which we can provide efficient recognition and layout algorithms. In Section 4.1 we focus on UER-RF drawings where the degree-3 vertices appear only on the external face; see Figure 3b. In Section 4.2 we take a further step towards the general case, by allowing internal degree-3 vertices, but requiring that the removal of the external cycle yields a collection of paths and cycles; see Figure 3c.

4.1 No internal degree-3 vertex

Differently from UER-USF drawings studied in Section 3, this setting allows internal degree-2 vertices. Also, we may have more than four external degree-2 vertices, which makes the

selection of the corners and the selection and drawing of the external cycle more challenging. We start with a property that extends Property 3, coming from the fact that each degree-4 (real- or crossing-) vertex has four 90° angles in its incident faces, whereas degree-2 vertices must have two 180° angles; see Figure 3b.

▶ **Property 4.** *A UER-RF drawing without internal degree-3 vertices contains disjoint vertical (horizontal) paths connecting each degree-3 vertex on the top (left) side with a degree-3 vertex on the bottom (right) side, possibly traversing internal degree-4 vertices. Furthermore, for each degree-2 vertex v in a vertical (horizontal) path there is a degree-2 vertex in the left and right (top and bottom) sides of the rectangle representing the external cycle, excluding the corners, having the same y-coordinate (x-coordinate) as v.*

In short, Property 4 states that every row (column) of the drawing either contains a single horizontal (vertical) path connecting the two opposite sides of the external rectangle or does not contain any horizontal (vertical) edge at all. In the following, we exploit this property to give several polynomial-time testing and construction algorithms for the problem, based on whether the corners, the external cycle, and/or the rotation system are part of the input.

▶ **Theorem 2.** *Let G be an n-vertex 4-graph. There exists a polynomial-time algorithm that tests whether G admits a UER-RF drawing with no internal degree-3 vertex. If such a drawing exists, the algorithm constructs one. Moreover, the algorithm can be adapted to preserve a given rotation system and/or to have a prescribed external cycle. The time complexity of the algorithm is:*

1. *$O(n)$ if the four corners are given;*
2. *$O(n)$ if the rotation system is prescribed;*
3. *$O(n)$ if G is a 3-graph and the external cycle is prescribed;*
4. *$O(n^3)$ if the external cycle is prescribed; and*
5. *$O(n^5)$ in the general case.*

As in Section 3, the algorithm supporting Theorem 2 is composed of two steps, the first one to select and draw the external cycle, and the the second one to draw the internal vertices. In Section 4.1.1 we describe several algorithms for the first step, depending on the specific setting. The algorithm to draw the internal vertices in the interior of a drawing of the external cycle is the same in all settings, and is described in Section 4.1.2.

4.1.1 Choosing and drawing the external face.

We start with the case in which the corners of the rectangle representing the external cycle are prescribed as part of the input (Item 1 of Theorem 2).

▶ **Lemma 3.** *Let G be an n-vertex 4-graph and let the corners of G be given. There exists an $O(n)$-time algorithm that constructs a constant number of rectangles such that if G admits a UER-RF drawing with no internal degree-3 vertex and with the prescribed corners, then the drawing of the external cycle coincides with one of these rectangles.*

Proof. We employ the same algorithm as the one in Section 3.1, using the four vertices prescribed in the input as the four corners. Namely, we remove the degree-4 vertices and smooth the degree-2 vertices different from the corners. This yields a graph G'' with all degree-3 vertices, except for the corners. We then guess all the possible orders of the four corners, add the edges between them according to this order, and test whether the resulting graph is triconnected and planar. If so, from the unique planar embedding, we read the at

most two possible cycles that can be used as external cycle in the desired UER-RF drawing. The only technical difference is that, in this case, the edges of G'' that are selected as chords may have been obtained by smoothing degree-2 vertices, while in Section 3.1 these edges had to be present as edges also in the original graph G, since in that case the smoothed degree-2 vertices could not be placed in the interior of the drawing. However, in the algorithm in Section 3.1 we did not exploit this property in this step, and deferred the discovery of this potential problem to Section 3.2, where the internal vertices are placed. As such, the algorithm can be used without modification also in this case. ◀

We then consider the case where the external cycle C is prescribed, but not the corners. In this case, by Condition (iii) of Property 1, deciding two adjacent corners allows us to infer the other two. Thus, one can guess $O(n^2)$ pairs of degree-2 vertices of C and get the corresponding $O(n^2)$ candidate rectangles (Item 4 of Theorem 2). In the following we show that, if we further have that G does not contain degree-4 vertices, then the rectangle corresponding to the given cycle (if any) is unique (Item 3 of Theorem 2).

▶ **Lemma 4.** *Let G be an n-vertex 3-graph and let the external cycle C of G be given. There exists an $O(n)$-time algorithm that constructs a rectangle Γ_C such that, if G admits a UER-RF drawing with no internal degree-3 vertex and with C as external cycle, then the drawing of the external face coincides with Γ_C.*

Proof. Since G has no degree-4 vertex and all its degree-3 vertices are on the external cycle C, we have that G consists of C and of a set $\mathcal{P}$ of disjoint paths connecting pairs of vertices of C. Our goal is to represent these paths as vertical or horizontal paths satisfying Property 4. We prove that there is only one possible drawing Γ_C of C in any UER-RF drawing of G whose external boundary coincides with C, up to rotation or a flip of the entire drawing, unless $G = C$.

Let v be a degree-3 vertex; refer to Figure 5a. By Property 1, v cannot be a corner of Γ_C. Let $P \in \mathcal{P}$ be the path incident to v, let u be the other endpoint of P, and let C_L and C_R be two paths along C between u and v. Observe that each of C_L and C_R must contain two of the corners of Γ_C, since u and v must lie on opposite sides of Γ_C, by Property 4. We assume, w.l.o.g., that there exist no two vertices of C_R that are connected by a path in $\mathcal{P}$, as otherwise we could select those two vertices as u and v. Let k be the number of degree-3 vertices in C_R. By the previous assumption, each of the k vertices is the endpoint of a path in $\mathcal{P}$ that crosses P in any UER-RF drawing of G.

Thus, by Property 4, the number of vertices on one side of Γ_C, say the right side, including the corners, must be equal to $|P| + k$, where $|P|$ denotes the number of vertices of P. Thus, the corners of Γ_C in C_R must be at distance $\frac{|C_R|-(|P|+k)}{2}$ from u and v, respectively, $|C_R|$ being the number of vertices of C_R. Also, the corners of Γ_C in C_L must be at distance $\frac{|C_L|-(|P|+k)}{2}$ from u and v. If one of the vertices chosen as corners has degree 3, we reject the instance by Property 1. Otherwise, we can realize Γ_C as a unit edge-length rectangle. ◀

We now consider the case in which the rotation system of G is prescribed, while the external cycle and the corners are not (Item 2 of Theorem 2). The strategy is to identify at most three cycles of G as potential external cycles, and then apply a strategy similar to Lemma 4 to find the corners. In particular, for degree-4 vertices we can use the rotation system to understand the direction of the paths passing through them. We have the following.

▶ **Lemma 5.** *Let G be an n-vertex 4-graph and let the rotation system $\mathcal{R}(G)$ of G be given. There exists an $O(n)$-time algorithm that constructs a constant number of rectangles such*

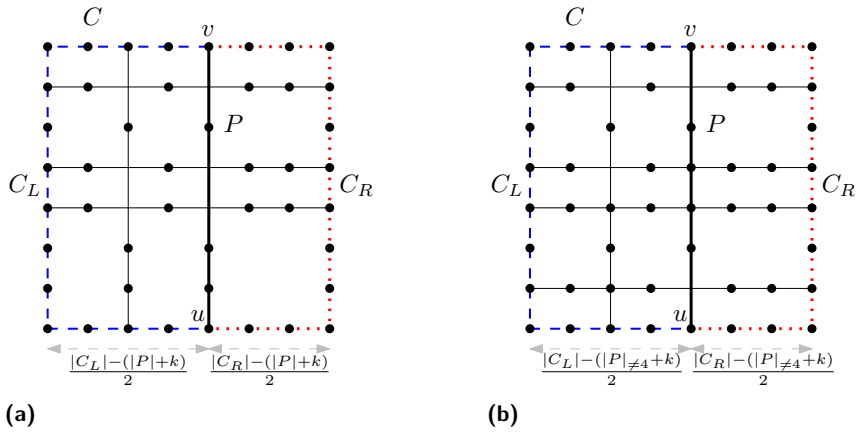


Figure 5 Illustrations for: (a) Lemma 4 ($k = 3$, $|P| = 5$); (b) Lemma 5 ($k = 4$, $|P| = 6$, $|P|_4 = 2$, $|P|_{\neq 4} = 4$). C_L is blue dashed, C_R is red dotted, P is bold.

that if G admits a UER-RF drawing with no internal degree-3 vertex and with rotation system $\mathcal{R}(G)$, then the drawing of the external face coincides with one of these rectangles.

Proof. First observe that, if G has no degree-3 vertex, then all the external vertices in any UER-RF drawing of G will have degree 2 in G, which is not possible unless G is a cycle (or the graph is disconnected). Assume then that G has at least one degree-3 vertex, and let v be such a vertex. By Property 1 and by the requirements of the lemma, v must be an external vertex distinct from the corners in any UER-RF drawing of G. In particular, one of the three faces incident to v must coincide with the external face. We guess each of the three faces and test them separately; this corresponds to guessing the two edges of C incident to v.

Let (v, w) and (v, z) be the two edges incident to v assigned to C, such that (v, w) follows (v, z) clockwise in $\mathcal{R}(v)$. If w has degree 4, we reject the current guess, by Property 1. If w has degree 2, we assign also its other incident edge to C and move on to analyzing this edge. If w has degree 3, we can use the rotation system of w to select the other edge incident to w that is in C; namely, we add to C the edge following (w, v) clockwise in $\mathcal{R}(w)$, and move on to analyzing this edge. When we reach a vertex whose edges have already been assigned to C, we check whether it coincides with v; if not, we reject the current guess as the constructed cycle C must be a simple cycle. Moreover, we check whether there are still degree-3 vertices in $G \setminus C$, in which case we reject the current guess. If we have not rejected the guess, we have obtained a simple cycle C containing all the degree-3 vertices of G.

In order to select the four corners among the vertices of C, we use the same strategy as in Lemma 4, namely we search for a set of paths $\mathcal{P}$ that connect pairs of vertices of C, and we use one of them to find the only selection of corners, if any, that complies with Property 4.

Note that, in this case, the paths in $\mathcal{P}$ are not necessarily disjoint, due to the presence of degree-4 vertices. However, we can assign to the same path each pair of edges that are not consecutive in the rotation system around a degree-4 vertex, since one of these paths will be drawn vertical and the other horizontal. This implies that the set $\mathcal{P}$ can still be constructed in linear time starting from the degree-3 vertices on C.

To compute the corners, let $P \in \mathcal{P}$ be the path between two vertices u and v such that one of the paths $C_R \subset C$ connecting u and v does not contain two vertices connected by another path in $\mathcal{P}$; refer to Figure 5b. Let k be the number of degree-3 vertices in C_R, let $|P|_4$ be the number of degree-4 vertices of P, and let $|P|_{\neq 4} = |P| - |P|_4$. By Property 4, each degree-2 vertex of P must be aligned with a degree-2 vertex of the right/left side of

Γ_C, and each degree-4 vertex of P must be aligned with a degree-3 vertex of the right/left side of Γ_C. Hence, if $k < |P|_4$ we reject the current guess; otherwise, the right side of Γ_C must contain $|P|_{\neq 4} + k$ vertices, so two of the corners must be at distance $\frac{|C_R|-(|P|_{\neq 4}+k)}{2}$ from u and v, respectively. The other two corners are computed by relying on Condition (iii) of Property 1. ◀

Finally, consider the general case in which no additional information is prescribed in the input (Item 5 of Theorem 2). In this case, we guess the $O(n^4)$ combinations of the four corners, and apply Lemma 3 to obtain $O(n^4)$ candidate rectangles in $O(n^5)$ time.

4.1.2 Drawing the internal vertices.

We now describe how to test whether one of the rectangles Γ_C constructed in Section 4.1.1 to represent the external cycle C can be completed to a UER-RF drawing of G, by placing the vertices of $G \setminus C$ in its interior. Recall that the vertices of $G \setminus C$ have degree either 2 or 4. Since the two edges incident to a degree-2 vertex must be drawn both vertical or both horizontal in a UER-RF drawing, we smooth such vertices and get a graph with all the internal vertices of degree 4. Thus, we can apply the same algorithm in Section 3.2 to compute the top/bottom/left/right neighbor of each degree-4 vertex. Recall that this assignment of neighbors is unique, if it exists. We then restore the degree-2 vertices and check if, consistently with Property 4, the resulting x- and y-coordinates of the degree-2 vertices coincide with those of the degree-2 vertices on the sides of Γ_C.

Proof of Theorem 2. The correctness follows from Property 4 and Lemmas 3–5, for the corresponding three cases, and from the exhaustive guesses in the other two. The time complexity comes from applying the linear-time algorithm described in Section 4.1.2 to each of the candidate rectangles constructed in the different cases, and since their number is bounded by a constant (Items 1-3 of Theorem 2) or by a polynomial in the number of vertices of the graph ($O(n^2)$ and $O(n^4)$ rectangles for Item 4 and 5 of Theorem 2, respectively).

4.2 Internal paths and cycles

We now consider the restricted scenario in which the input 4-graph G contains a cycle C such that $G \setminus C$ has vertex-degree at most two, i.e., it is a collection of paths and cycles (see, e.g., Figure 3c). We call G an *inner-2-graph with respect to* C. We only study the case when C is given as part of the input.

▶ **Theorem 6.** *Let G be an inner-2-graph with respect to a given cycle C. There exists a polynomial-time algorithm that tests whether G admits a UER-RF drawing whose external cycle coincides with C. If such a drawing exists, the algorithm constructs one. If the four corners are prescribed, the algorithm takes $O(n^2)$ time, otherwise it takes $O(n^4)$ time. Furthermore, the algorithm can be adapted to preserve a given rotation system.*

Algorithm. We first compute all rectangles that are candidate to represent the external boundary of the drawing. If the corners of C are prescribed in the input, then we have at most one rectangle respecting Property 1. Otherwise, as for Item 4 of Theorem 2, we guess $O(n^2)$ pairs of degree-2 vertices to be consecutive corners and infer the other two based on Condition (iii) of Property 1, thus obtaining $O(n^2)$ candidate rectangles.

Let Γ_C be one of the candidate rectangles computed in the previous step. In the rest of this section, we show how to place the vertices of $G \setminus C$ in the interior of Γ_C. Assume, w.l.o.g., that the bottom-left corner of Γ_C has coordinates $(0, 0)$. Let W and H be the maximum

x and y coordinates of Γ_C, respectively. We traverse the points (i, j) ($1 \leq i \leq W - 1$ and $1 \leq j \leq H - 1$) of the grid that lie internally to Γ_C from left to right and secondarily from top to bottom starting from the top-leftmost one, which has coordinates $(1, H - 1)$. Since we process the grid points from left to right and from top to bottom, the x-coordinate i increases and the y-coordinate j decreases. In the following, we call *placed vertices* those vertices whose coordinates have already been assigned by the algorithm (i.e., the vertices of C and those already placed at some internal grid points). When processing the point (i, j) we call *fixed vertices* the placed vertices with coordinates (i', j') such that either $i' < i$ or $i' = i$ and $j' > j$ (see Figure 6a).

During the traversal we maintain two arrays of vertices Up and Left of size W and H, respectively; see Figure 6a. Intuitively, they are used to extend the concept of top/left/bottom/right neighbors to all grid points, even if they do not contain a vertex. More specifically, when we process the point (i, j), let q be the rightmost fixed vertex such that $y(q) = j$; analogously, let r be the bottommost fixed vertex such that $x(r) = i$. If $\mathsf{Left}[j]$ stores a vertex q', then q' is adjacent to q and $y(q')$ will be equal to j whereas $x(q')$ has not been decided yet; in other words, it is already decided that q' will be horizontally aligned with q (and thus it is candidate to occupy point (i, j)). If $\mathsf{Left}[j]$ is null, then no neighbor of q will be assigned y-coordinate j; this implies that (i, j) cannot be occupied by a crossing. Analogously, if $\mathsf{Up}[i]$ stores a vertex r', then it is already decided that r' will be vertically aligned with r; if $\mathsf{Up}[i]$ is null, then no neighbor of r will be assigned x-coordinate i.

We initialize the array Left as follows. For $1 \leq j \leq H - 1$, let v_j be the vertex on the left side of Γ_C such that $y(v_j) = j$. If $\deg_G(v_j) = 2$, we set $\mathsf{Left}[j] = \mathsf{null}$. If $\deg_G(v_j) = 3$, we set $\mathsf{Left}[j] = u_j$, where u_j is the neighbor of v_j that is not on the left-side of Γ_C. The vertex u_j is either on the right side of Γ_C or it is an internal vertex of G. If $u_j \in C$ and $y(u_j) \neq y(v_j)$, we reject Γ_C. The Up array is initialized similarly, considering the vertices on the top side of Γ_C, and their neighbors that do not lie on the top side of Γ_C (if any).

We now explain how to process the generic point (i, j). We distinguish 5 cases depending on the values in the Left and Up arrays (see also Figures 6b–6f):

1. $\mathsf{Left}[j] = \mathsf{Up}[i]$ and they are both not null;
2. $\mathsf{Left}[j] \neq \mathsf{Up}[i]$ and they are both not null;
3. $\mathsf{Left}[j] \neq \mathsf{null}$ and $\mathsf{Up}[i] = \mathsf{null}$;
4. $\mathsf{Left}[j] = \mathsf{null}$ and $\mathsf{Up}[i] \neq \mathsf{null}$; and
5. $\mathsf{Left}[j] = \mathsf{Up}[i]$ and they are both null.

In Case 1, let v be the vertex stored in $\mathsf{Left}[j] = \mathsf{Up}[i]$. We map v to the point (i, j) and update the two arrays as explained below. In Case 2, we put a crossing in the point (i, j) and leave Left and Up unchanged. In Case 3, let $v = \mathsf{Left}[j]$; we map v to (i, j) and update the two arrays as described below. Similarly, in Case 4, let $v = \mathsf{Up}[i]$; we map v to (i, j) and update the two arrays as described below. Finally, in Case 5 the point (i, j) will be left unused (it will be internal to a face), and we leave Left and Up unchanged.

The updates of the values $\mathsf{Left}[j]$ and $\mathsf{Up}[i]$ in Cases 1, 3, and 4, are done as follows. Let U be the set of neighbors of v that are not fixed. If $|U| = 0$, we reject the instance, as either $\deg_G(v) = 2$ and its incident edges form a 270° angle (Case 1), or $\deg_G(v) = 1$ (Cases 3 and 4). Also, if $|U| \geq 3$, we reject the instance as in this case there is no possibility of placing all vertices in U. Thus it must be $|U| \in \{1, 2\}$; let u and u' be the two vertices of U, possibly with $u' = \mathsf{null}$ if $|U| = 1$. In this case, for each vertex u and u', either its coordinates are unassigned or it belongs to C. Suppose first that at least one of them, say u, belongs to C; if $x(v) \neq x(u)$ and $y(v) \neq y(u)$, we reject the instance because v and u cannot be horizontally or vertically aligned. If $x(v) = x(u)$ we set $\mathsf{Up}[i] = u$ and $\mathsf{Left}[j] = u'$; if $y(v) = y(u)$ we set

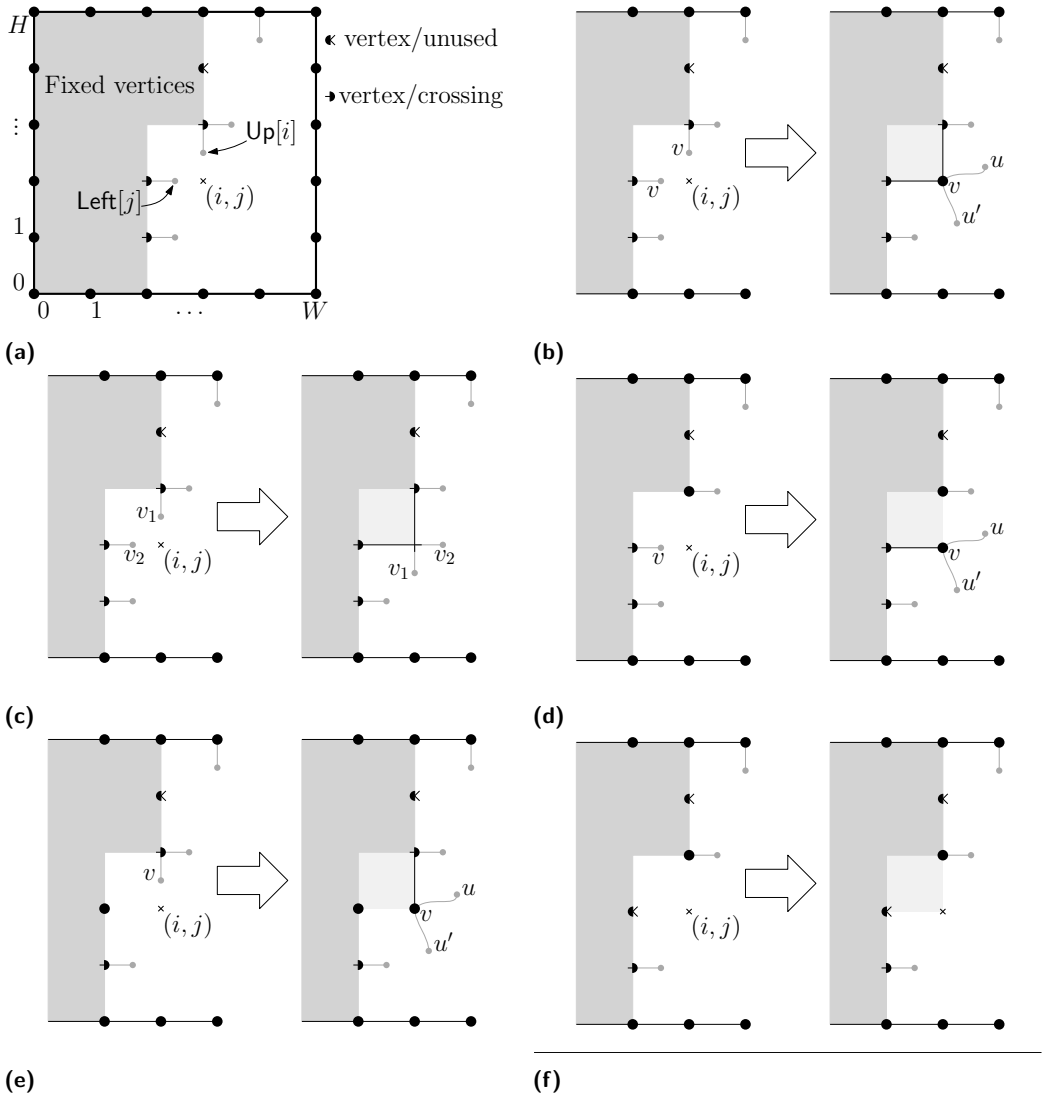


Figure 6 Illustration for Theorem 6. (a) Notation of the algorithm. (b)–(f) Processing the point (i, j): (b) Case 1; (c) Case 2; (d) Case 3; (e) Case 4; (f) Case 5.

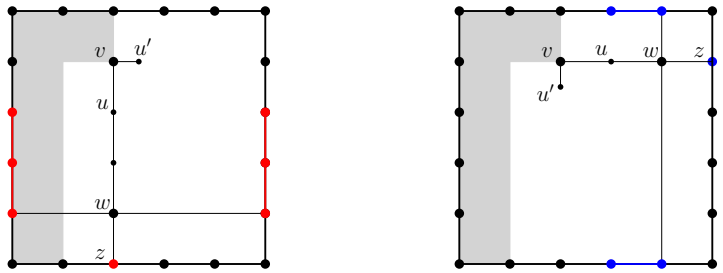


Figure 7 (a) If z coincides with one of the red points of Γ_C, then $x(u) = x(v)$. (b) If z coincides with one of the blue points of Γ_C, then $y(u) = y(v)$.

$\mathsf{Up}[i] = u'$ and $\mathsf{Left}[j] = u$. If u' exists and it is already placed, then we further check that its coordinates are consistent with its assignment to $\mathsf{Up}[i]$ or $\mathsf{Left}[j]$.

Suppose now that neither u nor u' is in C. If $\deg_G(u) \geq 3$ we set $w = u$; else we follow the edge incident to u distinct from (u, v) and continue traversing all degree-2 vertices until we reach a degree-3 vertex w. If w is fixed, we reject the instance, as either the traversed path can only be drawn with an angle of 270°, or u has to be on the left of v, contradicting the fact that it is not fixed. Hence w is non-fixed and either it belongs to C or its coordinates are unassigned. Assume first that $w \in C$. If $x(w) \neq x(v)$ and $y(w) \neq y(v)$ we reject the instance, as the path visited by the traversal must be either horizontal or vertical. Otherwise, if $x(w) = x(v)$ we set $\mathsf{Up}[i] = u$ and $\mathsf{Left}[j] = u'$. Similarly, if $y(w) = y(v)$ we set $\mathsf{Left}[j] = u$ and $\mathsf{Up}[i] = u'$. If the coordinates of w have not been assigned, then $w \in V(G) \setminus V(C)$. Since G is an inner-2-graph and $\deg_G(w) > 2$, at least one neighbor of w, call it z, belongs to C. If z lies on the left or right side of Γ_C, and $0 < y(z) < y(v)$ or $x(z) = x(v)$ and $y(z) = 0$, we set $\mathsf{Up}[i] = u$ and $\mathsf{Left}[j] = u'$ (see Figure 7a). If z lies on the top or bottom side of Γ_C, and $x(v) < x(z) < W$ or $x(z) = W$ and $y(z) = y(v)$, we set $\mathsf{Left}[j] = u$ and $\mathsf{Up}[i] = u'$ (see Figure 7b). If z does not lie in any of the described positions, we reject the instance, as z cannot share a coordinate with w, and thus the path from v to w cannot be drawn horizontally or vertically.

Once all the grid points have been considered, if there is a vertex of G with unassigned coordinates, we reject the instance, otherwise we have a drawing with external boundary Γ_C.

Correctness. We prove the correctness of the algorithm by proving that the following invariants hold when we process the grid point (i, j):

- **I1**: The drawing induced by the placed vertices is a UER-RF drawing with the only exception of one face which may not be rectangular; such a face (if it exists) is the one containing all the grid points still unprocessed.
- **I2**: If $\mathsf{Left}[j] \neq \mathsf{null}$, then the vertex $v = \mathsf{Left}[j]$ has y-coordinate equal to j; if $\mathsf{Left}[j] = \mathsf{null}$, none of the neighbors of the rightmost fixed vertex q with $y(q) = j$ has y-coordinate j.
- **I3**: If $\mathsf{Up}[i] \neq \mathsf{null}$, then the vertex $v = \mathsf{Up}[i]$ has x-coordinate equal to i; if $\mathsf{Up}[i] = \mathsf{null}$, none of the neighbors of the bottommost fixed vertex r with $x(r) = i$ has x-coordinate i.

The invariants hold before we process the first grid point internal to Γ_C. Namely, the drawing induced by the placed vertices is Γ_C and therefore it is a UER-RF drawing. Invariant **I2** holds because of the way we initialize the array Left. Namely, if $\deg_G(v_j) = 2$, we set $\mathsf{Left}[j] = \mathsf{null}$ and Invariant **I2** holds for $\mathsf{Left}[j]$ because all the neighbors of v_j belong to Γ_C; if $\deg_G(v_j) = 3$, we set $\mathsf{Left}[j] = u_j$ and Invariant **I2** holds for $\mathsf{Left}[j]$ because the only neighbor u_j of v_j that does not belong to Γ_C must be horizontally aligned with v_j. A symmetric

argument applies to Invariant **I3**.

Assume now that the invariants hold before processing the grid point (i, j). Because of invariants **I2** and **I3**, the decision taken by the algorithm about point (i, j) (Cases 1–5) is correct. Notice that each of the choices made in Cases 1–5 maintains Invariant **I1**. We now prove that the updates of the value $\mathsf{Left}[j]$ and $\mathsf{Up}[i]$ maintain the Invariants **I2** and **I3**. In Cases 2 and 5 $\mathsf{Left}[j]$ and $\mathsf{Up}[i]$ are not changed and the Invariants **I2** and **I3** still hold. Consider Cases 1, 3, and 4. In all the three cases a vertex v is placed at point (i, j) and $\mathsf{Left}[j]$ and $\mathsf{Up}[i]$ are updated looking at the non-fixed neighbors U of v. As already observed, if $|U| \in \{0, 3, 4\}$, the algorithm correctly rejects the instance. Otherwise, v has at most two non-fixed neighbors u and u' (with u' possibly equal to null). If one of the two, say u, belongs to C, then u and v must be aligned either horizontally, if they share the x-coordinate, or vertically, if they share the y-coordinate; in the first case we correctly set $\mathsf{Left}[j] = u$, while in the second case we correctly set $\mathsf{Up}[i] = u$ (if u and v do not share any coordinate we correctly reject the instance). If u' is not null, we correctly set $\mathsf{Up}[i] = u'$ if $\mathsf{Left}[j] = u$, and viceversa. Notice that, if u' is already placed we have to check that its coordinates allow the desired alignment; if not we reject the instance. If neither u nor u' belong to C, we search for a non-fixed vertex w whose degree is at least 3 such that v and w are connected by a chain of degree-2 non-fixed vertices, the first one being u (u and w may coincide, in which case the chain is empty). If w belongs to C, then v, w and all the vertices of the chain connecting v and w must be aligned either horizontally, if v and w share the y-coordinate, or vertically, if they share the x-coordinate; in the first case we correctly set $\mathsf{Left}[j] = u$, while in the second case we correctly set $\mathsf{Up}[i] = u$ (if w and v do not share any coordinate we correctly reject the instance). If w does not belong to C, then it will be adjacent to a vertex z that belongs to C. In each of the three cases described above (see also Figure 7) the position of z in Γ_C implies the vertical or horizontal alignment of v and u, or the rejection of the instance. In each case we correctly set the values of $\mathsf{Left}[j]$ and $\mathsf{Up}[i]$.

Time complexity. We conclude the proof by discussing the time complexity of our algorithm. We first compute all rectangles that are candidate to represent the external boundary of the drawing. As discussed above, if the set of corners is prescribed there is a unique candidate rectangle, whereas guessing the corners results in $O(n^2)$ candidate rectangles, which are computed by exploiting Condition (iii) of Property 1.

Consider now each of these rectangles Γ_C. As every point of the grid that lies internally to Γ_C is processed in one of the steps, we have $O(n^2)$ steps. In fact, the size of the grid can be quadratic with respect to the number of vertices of G. Each step in which a vertex is placed consists of constant-time operations on the arrays and on the vertex, plus a traversal of the graph that visits a path of degree-2 vertices that connects two vertices with higher degree. We claim that each vertex u is visited by at most four traversals. To prove the claim, it is sufficient to notice that each path of degree-2 vertices is traversed at most once. Thus, if $\deg_G(u) = 2$, u belongs to the path visited by a unique traversal, and if $\deg_G(u) \geq 3$, then there are at most two traversals having u as their starting point and two traversals having u as ending point. Thus, for a candidate rectangle Γ_C, the algorithm runs in $O(n^2)$ time.

Altogether, we conclude that if the four corners are given in the input, then the algorithm runs in $O(n^2)$ time; otherwise, the algorithm runs in $O(n^4)$ time.

5 Rectangular faces: general case

For the general case we describe an FPT algorithm in the number of degree-3 vertices (Theorem 8). To this aim, we first provide polynomial-time algorithms when the angles

around each degree-3 vertex are given in the input (Lemma 7). More formally, a *large-angle assignment* $\mathcal{A}(G)$ of a 4-graph G determines for each degree-3 vertex v of G a pair $\mathcal{A}(v) = \langle u, w \rangle$, such that $u, w \in N(v)$. We say that a UER-RF drawing Γ of G *realizes* a large-angle assignment $\mathcal{A}(G)$ if, for each degree-3 vertex v of G, with $\mathcal{A}(v) = \langle u, w \rangle$, we have in Γ that either $x(u) = x(w)$ or $y(u) = y(w)$, i.e. the angle delimited by the edges (u, v) and (v, w) is a 180°-degree angle. Note that $\mathcal{A}(G)$ does not imply a rotation system for G.

▶ **Lemma 7.** *Let G be a 4-graph with a large-angle assignment $\mathcal{A}(G)$. There exists a polynomial-time algorithm that tests whether G admits an angle-preserving UER-RF drawing. If such a drawing exists, the algorithm constructs one. Moreover, the algorithm can be adapted to preserve a given rotation system and/or to have a prescribed external cycle. The time complexity of the algorithm is:*

1. *$O(n^2)$ if the four corners are given;*
2. *$O(n^4)$ if the external cycle is prescribed;*
3. *$O(n^{4.5})$ in the general case.*

We present the algorithm supporting Lemma 7 in Sections 5.1 and 5.2, describing the choice and drawing of the external cycle and the placement of the internal vertices, respectively. By exploiting Lemma 7 we can obtain an FPT algorithm for the general case, parametrized by the number of degree-3 vertices, by guessing for each degree-3 vertex a pair of incident edges to form the large angle, and apply the algorithm of Lemma 7. We have the following.

▶ **Theorem 8.** *Let G be a 4-graph and let k be the number of degree-3 vertices of G. There exists an FPT algorithm, with parameter k, that tests whether G admits a UER-RF drawing. If such a drawing exists, the algorithm constructs one. Moreover, the algorithm can be adapted to preserve a given rotation system and/or to have a prescribed external cycle. The time complexity of the algorithm is:*

1. *$O(3^k n^2)$ if the four corners are given;*
2. *$O(3^k n^4)$ if the external cycle is prescribed;*
3. *$O(3^k n^{4.5})$ in the general case.*

5.1 Choosing and drawing the external face

Let G be a 4-graph with a large-angle assignment $\mathcal{A}(G)$. We show how to select a set of cycles of C and, for each of them, a set of rectangles representing it that bound the external face in any UER-RF drawing of G that realizes $\mathcal{A}(G)$.

We start with the case in which the four corners c_1, c_2, c_3, c_4 are given (Item 1 of Lemma 7). We show that, in this case, the external cycle C is unique. Let (u, v) be an edge that has been selected to be part of C, initially one of the edges incident to c_1. If $\deg_G(u) = 4$, we reject the instance (Figure 8a), by Property 1. If $\deg_G(u) = 2$, the other edge incident to u is also part of C, and we continue from there (Figure 8b). If $\deg_G(u) = 3$, the two edges forming the large angle must belong to C. Thus, if $v \notin \mathcal{A}(u)$ (Figure 8c), we reject the instance; otherwise (Figure 8d), the edge creating a large angle around u with (u, v) is in C, and we continue from it. If c_1 is reached, we check if the resulting cycle contains c_2, c_3, c_4 and if the lengths of the paths connecting them satisfy Condition (*iii*) of Property 1. If so, we can construct a unit edge-length grid rectangle representing C.

If the external cycle is prescribed, but not its corners (Item 2 of Lemma 7), we check for each degree-3 vertex if its two edges that are on C form the large angle, otherwise we reject the instance. Then, we guess all $O(n^2)$ pairs of adjacent corners, from which we infer the

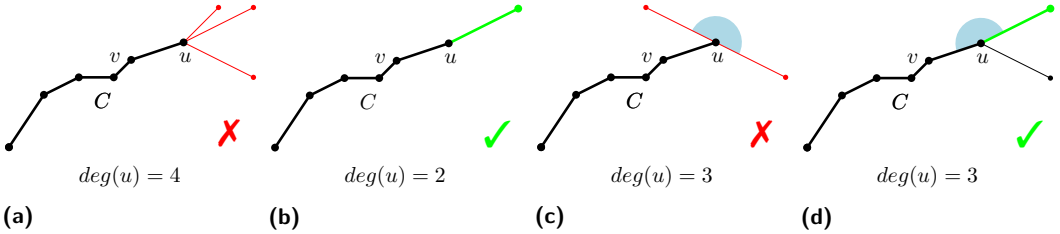


Figure 8 Candidate cycle construction: the green edge is the next edge of C.

other two corners using Condition (iii) of Property 1, and construct the corresponding $O(n^2)$ unit edge-length rectangles.

If neither the external cycle nor the corners are given (Item 3 of Lemma 7), note that the external cycle must contain at least one degree-3 vertex, as otherwise G would be a cycle or a non-connected graph. Thus, our strategy is to guess each degree-3 vertex u to be part of the external cycle and use the large-angle assignment to find the corresponding cycle, as in the proof of Item 1 of Lemma 7. In that proof, we have actually shown that there exists at most one candidate external cycle C that contains u. Hence, if we find such a cycle C, we do not need to guess any of the other degree-3 vertices that belong to C. We claim that a candidate external cycle C contains at least $\Omega(\sqrt{n})$ vertices, which implies that at the end of the process we have obtained at most $O(\sqrt{n})$ candidate external cycles. Together with Item 2 of Lemma 7, this implies $O(n^{2.5})$ different unit edge-length grid rectangles. We now prove the claim. Given a rectangle Γ_C, let h and w be the number of vertices different from the corners in the left and in the top side of Γ_C, respectively. We have that the number of grid points in the interior of Γ_C is $i = h \cdot w$, and so $n - |C| \leq i$ must hold. Thus $|C| \geq n - i$, and considering that the value of i is maximized when $h = w$, we have $|C| \geq n - h^2$. Finally, as $|C| = 2h + 2w + 4$ when $h = w$ we obtain $h^2 + 4h + 4 - n \geq 0$ which is solved for $h \geq \sqrt{n} - 2$. As $|C| > h$, we have that $|C| > \sqrt{n}$.

5.2 Drawing the internal vertices

From Section 5.1 we obtain either one or $O(n^2)$ or $O(n^{2.5})$ rectangles, respectively, in the three cases of Lemma 7. In the following we show how to place the vertices of $G \setminus C$ in the interior of one of these rectangles in $O(n^2)$ time.

The algorithm is similar to the one in Theorem 6. Namely, we analyze the points of the grid internally to Γ_C, moving from left to right and secondarily from top to bottom, maintaining two arrays Up and Left. Whenever a point (i, j) is visited, we decide if it is occupied by a vertex (real or dummy) or not (it is internal to a face), based on the values of $\mathsf{Up}[i]$ and $\mathsf{Left}[j]$, with the same five cases as in Theorem 6. Note that in Case 1 we also check if the placement of vertex $v = \mathsf{Up}[i] = \mathsf{Left}[j]$ is consistent with its large-angle assignment, if $\deg_G(v) = 3$.

The main difference with respect to Theorem 6 is in the update of arrays Up and Left after the placement of v, which is easier in this case when v has degree 3 due to the information deriving from its large-angle assignment. Namely, let (i, j) be the coordinates of v. Furthermore, let u and u' be the non-fixed neighbors of v, possibly $u' = \mathsf{null}$.

If $\deg_G(v) = 2$, we proceed as before: if $v = \mathsf{Left}[j]$ we set $\mathsf{Left}[j] = u$ and $\mathsf{Up}[i] = \mathsf{null}$, otherwise, we have that $v = \mathsf{Up}[i]$ and we set $\mathsf{Up}[i] = u$ and $\mathsf{Left}[j] = \mathsf{null}$.

If $\deg_G(v) = 3$, then let $\mathcal{A}(v) = \langle w, z \rangle$. We observe that exactly one of w and z is fixed, as pairs formed by the at most two fixed or by the at most two non-fixed neighbors of v

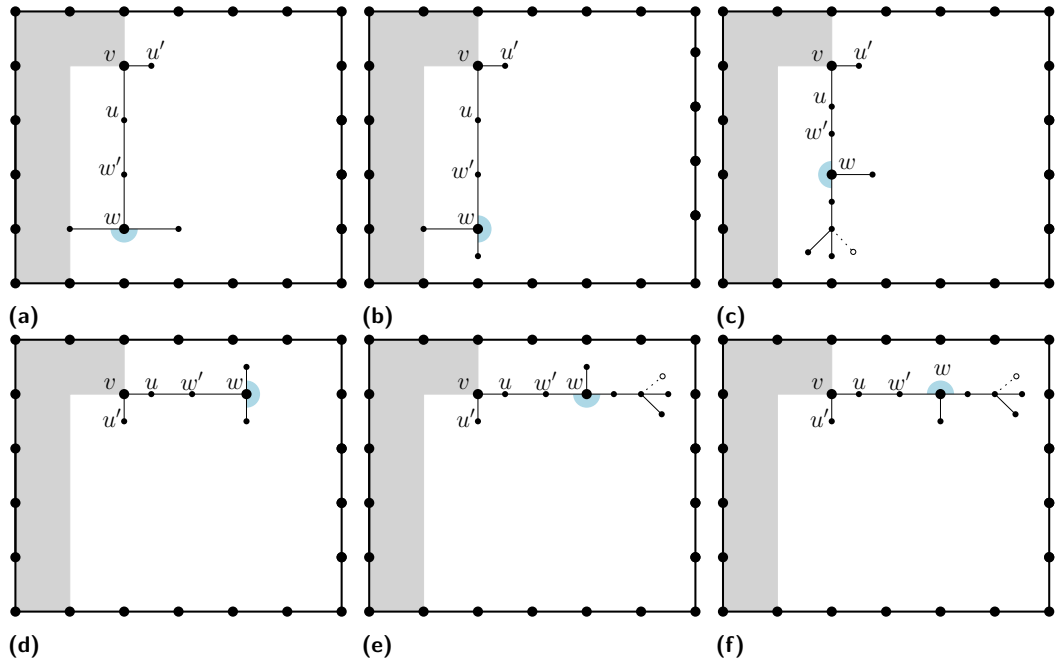


Figure 9 If $w' \notin \mathcal{A}(w)$, then if w has a fixed neighbor u is vertically aligned with v (a), otherwise it is horizontally aligned (d). If $w' \in \mathcal{A}(w)$, if it has a fixed neighbor, then it is vertically aligned with v (b). In the other cases (c)(e)(f), we continue the traversal along the large angle of w to find another vertex with degree 3 or 4.

cannot create a 180° angle. Therefore, we have that $u = w$, possibly after renaming, and we set $\mathsf{Up}[i]$ or $\mathsf{Left}[j]$ as u, so that u and z are aligned either vertically or horizontally, and the other will be set as u'.

If $\deg_G(v) = 4$, then we have no clear way of directly saying which of its two non-fixed neighbors should be vertically aligned with v, so we will exploit a traversal of the graph as in the algorithm of Theorem 6, starting from v and moving to u. Let w be a vertex encountered in this traversal. If $\deg_G(w) = 2$, we skip w and continue with its non-visited neighbor. If $\deg_G(w) = 4$, we observe that w can be vertically aligned with v if and only if it has exactly one fixed neighbor. Thus we can decide whether $\mathsf{Left}[j] = u$ and $\mathsf{Up}[i] = u'$ or vice versa by looking at the three neighbors of w, or reject as in Theorem 6. If $\deg_G(w) = 3$, we first check whether $w \in C$ and, if so, we either reject the instance or update Left and Up accordingly. Otherwise, we distinguish two cases based on whether the last visited vertex w' belongs to $\mathcal{A}(w)$ or not, see Figure 9.

If $w' \notin \mathcal{A}(w)$, we again have that w can be vertically aligned with v if and only if it has exactly one fixed neighbor, which allows us to make a decision as in the degree-4 case. Refer to Figure 9a for the case in which it has one fixed neighbor and so it can be vertically aligned, and to Figure 9d for the case in which it has no fixed neighbor and so it will be horizontally aligned; if w' has more than one fixed neighbor, we reject Γ_C.

If $w' \in \mathcal{A}(w)$, then again if w has a fixed neighbor or a neighbor on C, we can decide whether u is vertically or horizontally aligned with v, see Figure 9b. On the other hand, in this case, if w has no placed neighbor we can neither reject the instance nor make a decision, as both assignments are still possible; see Figures 9c, 9e, and 9f. In this case, we treat w as a degree-2 vertex, in the sense that we postpone the decision and move on to the edge incident to w that creates a large angle together with (w, w'). This concludes the description

of the algorithm.

The correctness and the time complexity can be proved similarly to Theorem 6. The decisions based on large-angle assignments have been justified during the description of the algorithm and can be performed in constant time per vertex. It is important to note that each vertex is visited in a constant number of traversals, as for each of them we either make a final decision or we find a unique edge to follow in the next step of the traversal. The $O(n^2)$ time complexity is thus determined by the size of the grid. Overall, the time complexity of the algorithm descends from the number of candidate rectangles (either one, $O(n^2)$, or $O(n^{2.5})$) and the $O(n^2)$ time to test them.

6 Open problems

We conclude with open problems that arise from our results. We have shown that in the context of UER-RF drawings, degree-3 vertices play a special role, as we provided polynomial-time algorithms when their arrangement in the graph has a special structure or when their number is limited; in fact, we also presented an FPT algorithm parameterized by the number of such vertices. The complexity of the problem in the general case remains open, as well as the question of whether one can improve our super-linear time algorithms. Additionally, it would be interesting to extend our drawing convention to also handle vertices of high degree (larger than 4). Finally, an extension to 3D drawings could be considered, requiring that the internal regions have a parallelepipedal shape.

References

1 Zachary Abel, Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Jayson Lynch, and Tao B. Schardl. Who needs crossings? Hardness of plane graph rigidity. In *32nd Int. Symp. on Computational Geometry (SoCG '16)*, volume 51 of *LIPIcs*, pages 3:1–3:15, 2016. doi:10.4230/LIPIcs.SoCG.2016.3.

2 Carlos Alegría, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Fabrizio Grosso, and Maurizio Patrignani. Unit-length rectangular drawings of graphs. In *30th Int. Symp. on Graph Drawing and Network Visualization (GD'22)*, volume 13764 of *LNCS*, pages 127–143. Springer, 2022. doi:10.1007/978-3-031-22203-0_10.

3 Carlos Alegría, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Fabrizio Grosso, and Maurizio Patrignani. Unit-length rectangular drawings of graphs. *J. Graph Algorithms Appl.*, 28(1):403–437, 2024. doi:10.7155/JGAA.V28I1.2996.

4 Patrizio Angelini, Michael A. Bekos, Julia Katheder, Michael Kaufmann, Maximilian Pfister, and Torsten Ueckerdt. Axis-parallel right angle crossing graphs. In *31st Ann. Eur. Symp. on Algorithms (ESA'23)*, volume 274 of *LIPIcs*, pages 9:1–9:15, 2023. doi:10.4230/LIPICS.ESA.2023.9.

5 Sandeep N. Bhatt and Stavros S. Cosmadakis. The complexity of minimizing wire lengths in VLSI layouts. *Inf. Process. Lett.*, 25(4):263–267, 1987. doi:10.1016/0020-0190(87)90173-6.

6 Sergio Cabello, Erik D. Demaine, and Günter Rote. Planar embeddings of graphs with specified edge lengths. In Giuseppe Liotta, editor, *Graph Drawing, 11th International Symposium, GD 2003*, volume 2912 of *LNCS*, pages 283–294. Springer, 2003. doi:10.1007/978-3-540-24595-7_26.

7 Sergio Cabello, Erik D. Demaine, and Günter Rote. Planar embeddings of graphs with specified edge lengths. *J. Graph Algorithms Appl.*, 11(1):259–276, 2007. doi:10.7155/JGAA.00145.

8 Sabine Cornelsen and Andreas Karrenbauer. Accelerated bend minimization. In Marc van Kreveld and Bettina Speckmann, editors, *Graph Drawing*, pages 111–122, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

9 Sabine Cornelsen and Andreas Karrenbauer. Accelerated bend minimization. *J. Graph Algorithms Appl.*, 16(3):635–650, 2012. URL: https://doi.org/10.7155/jgaa.00265, doi:10.7155/JGAA.00265.

10 Giuseppe Di Battista and Walter Didimo. GDToolkit. In Roberto Tamassia, editor, *Handbook on Graph Drawing and Visualization*, pages 571–597. Chapman and Hall/CRC, 2013.

11 Walter Didimo. Right angle crossing drawings of graphs. In *Beyond Planar Graphs*, pages 149–169. Springer, 2020. doi:10.1007/978-981-15-6533-5_9.

12 Walter Didimo, Giuseppe Liotta, and Fabrizio Montecchiani. A survey on graph drawing beyond planarity. *ACM Comput. Surv.*, 52(1):4:1–4:37, 2019. doi:10.1145/3301281.

13 Peter Eades and Nicholas C. Wormald. Fixed edge-length graph drawing is NP-hard. *Disc. Appl. Math.*, 28(2):111–134, 1990. doi:10.1016/0166-218X(90)90110-X.

14 Stefan Felsner. Rectangle and square representations of planar graphs. In János Pach, editor, *Thirty Essays on Geometric Graph Theory*, pages 213–248. Springer New York, New York, NY, 2013. doi:10.1007/978-1-4614-0110-0_12.

15 Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. In Roberto Tamassia and Ioannis G. Tollis, editors, *DIMACS International Workshop (GD'94)*, volume 894 of *Lecture Notes in Computer Science*, pages 286–297. Springer, 1994. doi:10.1007/3-540-58950-3_384.

16 Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.*, 31(2):601–625, 2001. doi:10.1137/S0097539794277123.

17 Angelo Gregori. Unit-length embedding of binary trees on a square grid. *Inf. Process. Lett.*, 31(4):167–173, 1989. doi:10.1016/0020-0190(89)90118-X.

18 Siddharth Gupta, Guy Sa'ar, and Meirav Zehavi. Grid recognition: Classical and parameterized computational perspectives. *J. Comput. Syst. Sci.*, 136:17–62, 2023. doi:10.1016/J.JCSS.2023.02.008.

19 Seok-Hee Hong. Beyond planar graphs: Introduction. In *Beyond Planar Graphs*, pages 1–9. Springer, 2020. doi:10.1007/978-981-15-6533-5_1.

20 John E. Hopcroft and Robert Endre Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973. doi:10.1137/0202012.

21 John E. Hopcroft and Robert Endre Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974. doi:10.1145/321850.321852.

22 Takao Nishizeki and Md. Saidur Rahman. *Planar Graph Drawing*, volume 12 of *Lecture Notes Series on Computing*. World Scientific, 2004. doi:10.1142/5648.

23 Takao Nishizeki and Md. Saidur Rahman. Rectangular drawing algorithms. In Roberto Tamassia, editor, *Handbook on Graph Drawing and Visualization*, pages 317–348. Chapman and Hall/CRC, 2013.

24 Maurizio Patrignani. Planarity testing and embedding. In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*, pages 1–42. Chapman and Hall/CRC, 2013.

25 Md. Saidur Rahman, Takao Nishizeki, and Shubhashis Ghosh. Rectangular drawings of planar graphs. In Stephen G. Kobourov and Michael T. Goodrich, editors, *Graph Drawing, 10th International Symposium, GD 2002*, volume 2528 of *Lecture Notes in Computer Science*, pages 244–255. Springer, 2002. doi:10.1007/3-540-36151-0_23.

26 Md. Saidur Rahman, Takao Nishizeki, and Shubhashis Ghosh. Rectangular drawings of planar graphs. *J. Algorithms*, 50(1):62–78, 2004. doi:10.1016/S0196-6774(03)00126-3.

27 Marcus Schaefer. The graph crossing number and its variants: A survey. *The Electronic Journal of Combinatorics*, 1000:DS21: May 17, 2024, Apr. 2013. URL: https://www.combinatorics.org/ojs/index.php/eljc/article/view/DS21, doi:10.37236/2713.

28 Marcus Schaefer. Realizability of graphs and linkages. In János Pach, editor, *Thirty Essays on Geometric Graph Theory*, pages 461–482. Springer New York, New York, NY, 2013. doi:10.1007/978-1-4614-0110-0_24.

29 Roberto Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987. doi:10.1137/0216030.