Drawn Tree Decomposition: New Approach for Graph Drawing Problems

Siddharth Gupta
BITS Pilani, Goa Campus, India

Guy Sa'ar
Ben Gurion University of the Negev, Beersheba, Israel

Meirav Zehavi
Ben Gurion University of the Negev, Beersheba, Israel

Abstract

Over the past decade, we witness an increasing amount of interest in the design of exact exponential-time and parameterized algorithms for problems in Graph Drawing. Unfortunately, we still lack knowledge of general methods to develop such algorithms. An even more serious issue is that, here, "standard" parameters very often yield intractability. In particular, for the most common structural parameter, namely, treewidth, we frequently observe NP-hardness already when the input graphs are restricted to have constant (often, being just 1 or 2) treewidth.

Our work deals with both drawbacks simultaneously. We introduce a novel form of tree decomposition that, roughly speaking, does not decompose (only) a graph, but an entire drawing. As such, its bags and separators are of geometric (rather than only combinatorial) nature. While the corresponding parameter – like treewidth – can be arbitrarily smaller than the height (and width) of the drawing, we show that – unlike treewidth – it gives rise to efficient algorithms. Specifically, we get slice-wise polynomial (XP) time algorithms parameterized by our parameter. We present a general scheme for the design of such algorithms, and apply it to several central problems in Graph Drawing, including the recognition of grid graphs, minimization of crossings and bends, and compaction. Other than for the class of problems we discussed in the paper, we believe that our decomposition and scheme are of independent interest and can be further extended or generalized to suit even a wider class of problems. Additionally, we discuss classes of drawings where our parameter is bounded by $\mathcal{O}(\sqrt{n})$ (where n is the number of vertices of the graph), yielding subexponential-time algorithms. Lastly, we prove which relations exist between drawn treewidth and other width measures, including treewidth, pathwidth, (dual) carving-width and embedded-width.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Human-centered computing → Graph drawings; Theory of computation → Computational geometry

Keywords and phrases Graph Drawing, Parameterized Complexity, Tree decomposition

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.23

Related Version *Full Version*: `https://arxiv.org/abs/2310.05471` [33]

Funding *Siddharth Gupta*: Supported by Engineering and Physical Sciences Research Council (EPSRC) grant EP/V007793/1.
Guy Sa'ar: Supported in part by the Israeli Smart Transportation Research Center and by the Lynne and William Frankel Center for Computing Science at Ben-Gurion University.
Meirav Zehavi: Supported by the European Research Council (ERC) grant titled PARAPATH.

18th International Symposium on Parameterized and Exact Computation (IPEC 2023).
Editors: Neeldhara Misra and Magnus Wahlström; Article No. 23; pp. 23:1–23:22
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Over the past decade, we witness an increasing amount of interest in the design of exact exponential-time and parameterized algorithms for problems in Graph Drawing. For a few illustrative examples, let us mention that this includes studies of crossing minimization [32, 37, 38], recognition of planar graph families such as upward planarity testing [16, 35] and grid graph recognition [34], as well as recognition of beyond planar graph families [4], turn-minimization [27], linear layouts such as books embeddings [5, 8], clustered planarity and hybrid planarity [19, 40, 20], and bend minimization [24, 22]. For more information on recent progress on these and other topics, we refer to the report [28] and surveys such as [48]. Unfortunately, still, we have very limited knowledge of general methods to develop exact exponential-time and parameterized algorithms for problems in Graph Drawing.

An even more serious issue is that, for Graph Drawing problems, "standard" parameters very often yield intractability. In particular, for the most common structural parameter, namely, treewidth,[1] we frequently observe NP-hardness already when the input graphs are restricted to have constant (often, being just 1 or 2) treewidth. The same result holds even for the larger parameter pathwidth. For example, Grid Recognition is NP-hard on graphs of treewidth 1 (being trees) [7] or pathwidth 2 [34], Orthogonal Compaction is NP-hard even on cycles [26] and hence on graphs of pathwidth (and treewidth) 2, Min-Area Planar Straight-line Drawing is NP-complete on outerplanar graphs and hence on graphs of treewidth 2 [9, 39], and Grid Upward Drawing is NP-complete on graphs of treewidth 1 (being trees) [1, 10]. In light of this, we must seek parameterizations that are larger (or incomparable) to treewidth. Due to the nature of the problems at hand, it is natural to seek parameters of geometric flavors. Here, perhaps, the first choice that comes to mind is the *height* (or, rather, the minimum among the height and width) of the sought (or given) drawing. In particular, we can easily observe that this parameter for planar orthogonal grid drawings is bounded by $\Omega(\mathsf{tw})$, where tw is the treewidth of the drawn graph, and that it gives rise to the use of dynamic programming. However, denoting the number of vertices by n, we can also easily observe that this parameter can be as large as $\Omega(n)$ for ridiculously simple planar orthogonal grid drawings (and graphs)! For example, consider the path drawn in Figure 1a – here, already, both height and width are equal to (roughly) $n/2$.

Our work deals with both drawbacks mentioned above simultaneously. We introduce a novel form of tree decomposition that, roughly speaking, does not decompose (only) a graph, but an entire drawing. As such, its bags and separators are of geometric (rather than only combinatorial) nature. We further discuss this concept (still informally but in more detail) in Section 1.1 ahead. While the corresponding parameter – like treewidth – can be arbitrarily smaller than the height (and width) of the drawing (e.g., for the aforementioned example in Figure 1a, our parameter is a fixed constant), we show that – unlike treewidth – it gives rise to efficient (that is, XP) algorithms. Specifically, we present a general scheme for the design of such algorithms (described in Section 1.3), and apply it to several central problems in Graph Drawing, including the recognition of grid graphs, minimization of crossings and bends, and compaction (see Section 1.4). We believe that our new concept of geometric tree decomposition is interesting on its own, and exploring the connections between it and notions concerning (classical) tree decompositions is a promising research direction. Furthermore, we believe that this concept and our scheme can be further extended or generalized to be applicable to problems other than those discussed in this paper. Due to lack of space, several concepts and proofs are deferred to the full version of this paper [33].

[1] Definitions of standard terms and notations used in the Introduction can be found in Section A.

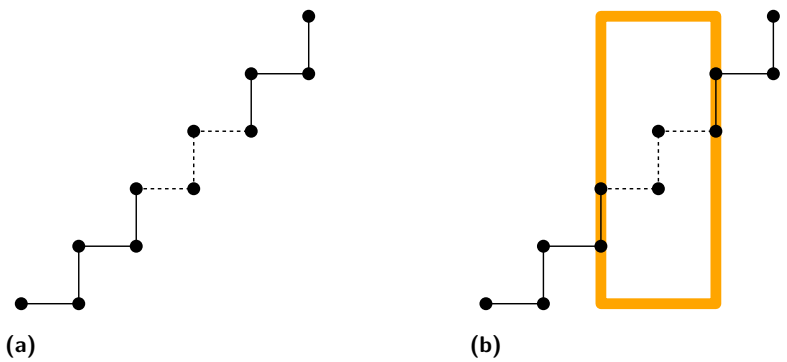


Figure 1 (a) A drawing of a path on n vertices with height and width $(n-1)/2$. However, the drawn treewidth is 16. (b) An illustration of a frame shown in orange with width 16.

1.1 The Concept of Drawn Tree Decomposition

Here, we discuss (informally) our main conceptual contribution: the introduction and study of the concepts of *drawn tree decomposition* and *drawn treewidth*, which we believe to be of independent interest. Then, in Section 1.2, we compare our parameter with several seemingly related graph parameters. Later, in Sections 1.3 and 1.4, we discuss our main technical contribution (which has been our initial motivation for these concepts): our general algorithmic scheme and its applications to problems in Graph Drawing. Our focus is on a class of rather general drawings of graphs on the Euclidean plane (allowing drawings of edges to have both crossings and bends, as well as to consist of segments that are not necessarily parallel to the axes), called *polyline grid drawings*. Roughly speaking, a polyline grid drawing d of a graph G is a mapping of the vertices of G to distinct grid points (being points of the form (i, j) where $i, j \in \mathbb{Z}$) and edges to *straight-line paths* between their endpoints. That is, the drawing of an edge is a simple curve that is the concatenation of straight-line segments (e.g, see Figure 11e in Section A.2). Towards the (informal) definition of a drawn tree decomposition ahead, we first introduce three critical terms: *frame*, *cutter*, and *rectangular*.

Frame. A *frame* is, simply, a straight-line cycle (defined analogously to a straight-line path above) whose segments are axis-parallel (see the orange polygon in Figure 2). In other words, it is a simple rectilinear polygon whose vertices lie on grid points.

For the definition of the width of our decomposition (presented later), we define the *width of a frame*. Roughly speaking, the width of a frame f, denoted by $\mathsf{width}(f)$, is the sum of measures of the complexities of *(i)* the frame itself, and *(ii)* the "way" in which the drawing "traverses" the frame. For *(i)*, we simply count the number of vertices of the frame (ignoring "superfluous" vertices, being those where the angle between incident edges is of 180 degrees). For *(ii)*, we regard the drawings of vertices and edges separately (and sum up the two corresponding numbers). Specifically, for vertices, we simple count the number of vertices drawn on the frame. However, for edges, the measure is somewhat more complex, based on the notion of *turning points* (defined immediately); for each edge, we count the number of its turning points on the frame, and, then, the measure is the sum (over all edges) of these counters. We remark that some points on the plane might be counted multiple times – at the extreme case, the same point might be *(a)* a vertex of the frame, *(b)* a point on which a vertex of the graph is drawn, and *(c)* a turning point for one (or more) edges. We find this multi-count to be justified: the more complicated the frame and the drawing are at a certain point, the more that point "contributes" to the complexity of the measure.

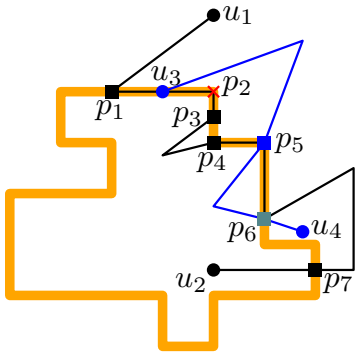


Figure 2 The turning points of the black edge $e = \{u_1, u_2\}$ in the orange frame f are $(p_1, e), (p_3, e), (p_4, e)$, (p_6, e) and (p_7, e). Note that, (p_2, e) and (p_5, e) are not turning points in f. Similarly, the turning points of the blue edge $e' = \{u_3, u_4\}$ in the frame f are $(d(u_3), e'), (p_5, e')$ and (p_6, e'). Note that (p_5, e') is a turning point in f, but (p_5, e) is not a turning point in f.

Now, let us define the notion of a turning point. For this purpose, consider some edge $e = \{u, v\}$ of the graph and some point p on the frame. Then, roughly speaking, we refer to (p, e) as a turning point if, when we traverse the drawing of e from u to v or from v to u, we encounter p, and "immediately" before this encounter, we were in the strict interior or exterior of the frame. Additionally, we refer to (p, e) as a turning point if u or v themselves are drawn on p. An illustrative example is given in Figure 2.

For an example of the definition of the width of a frame, we refer to Figure 1b. Here, the frame itself (being a rectangle) consists of exactly 4 vertices. Second, the path contains exactly 4 vertices that are drawn on the frame. Third, every point on which one of these vertices, say, v, is drawn is a turning point of 2 edges, being the two edges incident to v. So, the width of the frame is $4 + 4 + 4 \cdot 2 = 16$.

Cutter. A *cutter of a frame* is, simply, a straight-line path whose segments are axis-parallel and which intersects the frame in exactly two points, which are the endpoints of the cutter. Later, we discuss the "futility" of two simpler definitions for a cutter. The utility of a cutter of a frame f is, as its name suggests, in "cutting" f into (exactly) two frames f_1 and f_2. Roughly speaking, we obtain one of f_1 and f_2 by the concatenation of the cutter with one path among the two subpaths of f between the endpoints of the cutter, and we obtain the other of f_1 and f_2 by the concatenation of the cutter with the other path among the two subpaths of f between the endpoints of the cutter. For more intuition, we refer the reader to Figure 3.

Rectangular. For the sake of intuition, the construction of a drawn tree decomposition may be thought of as a recursive process where, for a given frame, we compute a cutter that cuts it into two, and then proceed (recursively) with each of these two resulting frames. Then, two questions arise: What is the initial frame, and when does this process terminate? For the first question, the answer is simply the *rectangular* of the drawing (defined immediately). For the second question, the answer is even simpler – we stop when the current frame does not contain any grid point in its strict interior. Roughly speaking, the rectangular of a drawing is the (unique) frame whose interior is minimized among all frames whose "shape" is a rectangle and which contain the given drawing in their strict interior (see Figure 4).

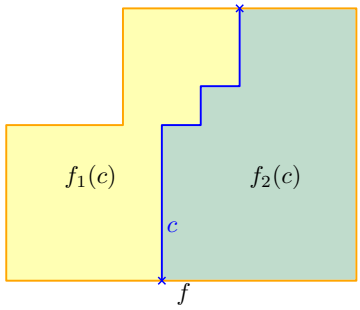


Figure 3 An illustration for a cutter c, shown in blue, of a frame f, shown in orange, and its associated frames $f_1(c)$ and $f_2(c)$.

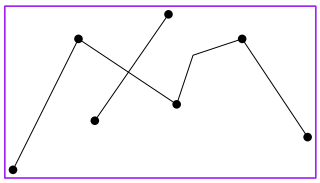
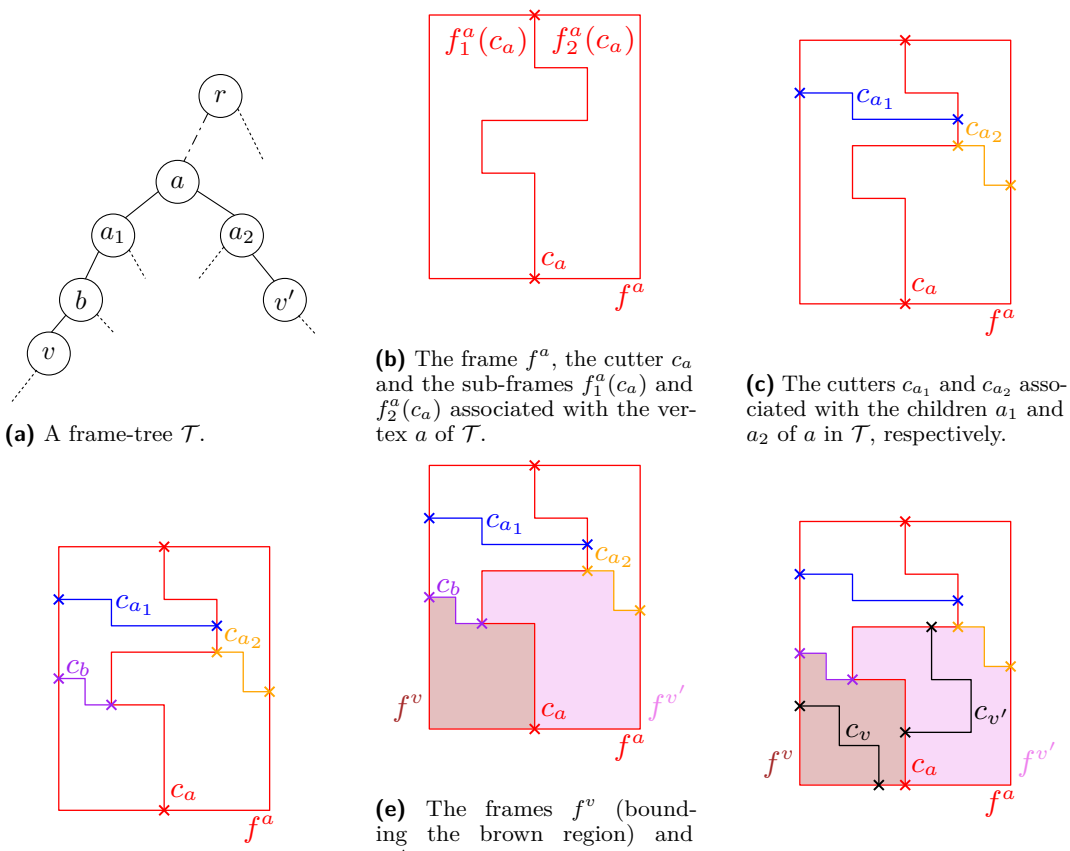


Figure 4 The frame shown in purple is R_d where d is the drawing inside the frame.

Drawn Tree Decomposition. At the heart of the concept of a drawn tree decomposition, lies our definition of a *frame-tree* (abbreviation for *tree of frames*). Informally, for a graph G and a polyline grid drawing d of G, a frame-tree is a pair $(\mathcal{T}, \alpha)$ where $\mathcal{T}$ is a binary rooted tree and α maps each vertex of $\mathcal{T}$ to a frame, such that: *(i)* the root is mapped to the rectangular of d; *(ii)* for every internal vertex v of $\mathcal{T}$, there exists a (unique) cutter c_v of $\alpha(v)$ so that the frames mapped to the children of v are those obtained by cutting $\alpha(v)$ by c_v; *(iii)* the leaves of $\mathcal{T}$ (and none of the internal vertices of $\mathcal{T}$) are mapped to frames whose strict interior does not contain any grid point. For an illustrative example, see Figure 5.

Now, for the definition of a drawn tree decomposition, we consider a frame-tree $(\mathcal{T}, \alpha)$. Then, we "enrich" the frame-tree by the introduction of an additional mapping, β, from the vertex set of $\mathcal{T}$ to subsets of vertices of G. In particular, we define β so that we can: *(P1)* prove that $(\mathcal{T}, \beta)$ is a tree decomposition (this proof is slightly technical, based on case analysis); *(P2)* prove that, for every vertex v of $\mathcal{T}$, $|\beta(v)|$ is at most twice the sum of the widths of the frames of v and its two children (if they exist). For the definition of β, we (next) define the *set of vertices associated with a frame*, and the *set of vertices associated with a cutter* of a frame. Then, for a vertex v of $\mathcal{T}$, $\beta(v)$ is simply the union of the set of vertices associated with $\alpha(v)$, and the set of vertices associated with the cutter c_v of $\alpha(v)$. Correspondingly, the triple $(\mathcal{T}, \alpha, \beta)$ is a drawn tree decomposition.

So, consider a graph G, a polyline grid drawing d of G, a frame f and a cutter c of f. Then, the *set of vertices associated with* f is the union of the set of vertices of G that d draws on f and the set of endpoints of edges of G whose drawing (by d) is *separated* by f – that is, edges having one endpoint in the strict interior of f and the other endpoint in the strict exterior of f (see Figure 6a). Similarly, the *set of vertices associated with* c is the union of the set of vertices of G that d draws on c and the set of endpoints of edges of G whose drawing (by d) is *separated* by c – that is, edges having one endpoint in the strict interior of one of the frames obtained by cutting f by c, and the other endpoint in the strict interior of the other frame obtained by cutting f by c (see Figure 6b).



(a) A frame-tree $\mathcal{T}$.

(b) The frame f^a, the cutter c_a and the sub-frames $f_1^a(c_a)$ and $f_2^a(c_a)$ associated with the vertex a of $\mathcal{T}$.

(c) The cutters c_{a_1} and c_{a_2} associated with the children a_1 and a_2 of a in $\mathcal{T}$, respectively.

(d) The cutter c_b associated with the child b of a_1 in $\mathcal{T}$.

(e) The frames f^v (bounding the brown region) and $f^{v'}$ (bounding the pink region) associated with the vertices v and v' of $\mathcal{T}$, respectively.

(f) The cutters c_v and $c_{v'}$ associated with the vertices v and v' in $\mathcal{T}$, respectively.

Figure 5 Example of frames and cutters of a frame-tree. For clarity, the polyline grid drawing is not shown.

Drawn Treewidth. The *width* of a drawn tree decomposition $(\mathcal{T} = (V_T, E_T), \alpha, \beta)$ is the maximum width of its frames, that is, $\max_{v \in V_T} \mathsf{width}(\alpha(v))$. Accordingly, the drawn treewidth of a polyline grid drawing d of a graph G is the minimum width of a drawn tree decomposition of d. Notably, due to *(P1)* and *(P2)* mentioned above, we can easily conclude that the treewidth of G is at most 6 times its drawn treewidth.

We remark that the usage of frames bears similarity to that of *cycle separators of planar graphs* (being a central player in proofs of the planar separator theorem; see, e.g., [3, 42]). However, the corresponding widths (drawn treewidth versus treewidth) can be critically different: While treewidth is bounded from above by the order of drawn treewidth, we have already pointed out that for various problems where treewidth yields intractability, drawn treewidth does not – this, of course, implies that treewidth can, often, be arbitrarily smaller than drawn treewidth; for a concrete example, see Figure 7. Further, treewidth depends only on the graph, while drawn treewidth depends (as desired) on the drawing; for example, notice that Figures 1a and 7a depict the same graph, but the corresponding drawings have radically different drawn treewidths.

Besides its above-mentioned relation to treewidth, drawn treewidth for planar orthogonal grid drawings can also be related to height (and width). On the one hand, we prove the desirable property that – like treewidth – drawn treewidth is bounded from above by the order

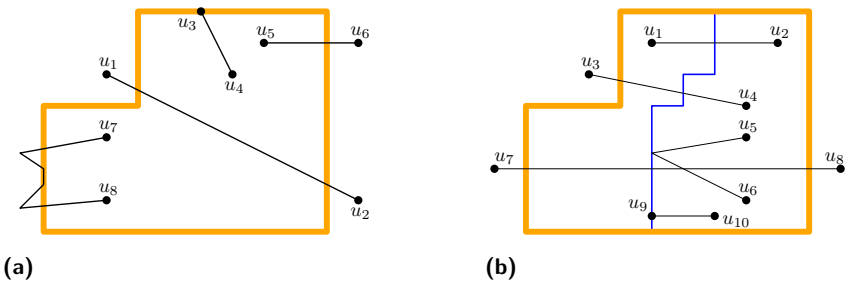


Figure 6 Example of vertices associated with a frame and a cutter. (a) The edge $\{u_5, u_6\}$ is the only edge separated by the orange frame. The vertices associated with the orange frame are u_3, u_5 and u_6. (b) The vertices associated with the blue cutter of the orange frame are u_1, u_2 and u_9.

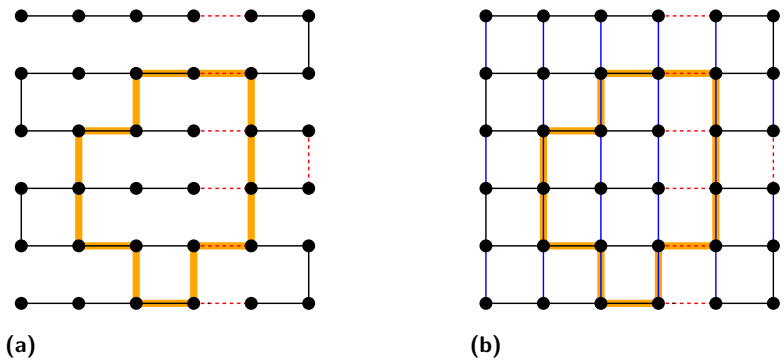


Figure 7 (a) A path P on n vertices and a frame f shown in orange. (b) A grid graph G on the same set of vertices and the frame f shown in orange. Consider a frame, say f, in P with width w. Observe that f is also a frame in G. Moreover, the width of f in G is at most $3w$ as every vertex has exactly 2 more edges in G compared to P so the vertex may be counted 2 more times in the width of f in G as the turning points of those 2 extra edges. As treewidth is a lower bound for drawn treewidth and the treewidth of a grid graph is $\sqrt{n}$, the drawn treewidth of P is $\Omega(\sqrt{n})$ (while its treewidth is 1).

of the minimum among the height and width of the drawing. Notably, various central graph width measures do *not* have this property. For example, one of the most commonly used relaxations of pathwidth is *treedepth* (see, e.g., [18] for information on treedepth); however, the treedepth of an n-vertex path is $\lceil \log_2(n+1) \rceil$, while it can be easily drawn so that the height (or, symmetrically, width) of the drawing is 1. On the other hand, we have already observed that the drawn treewidth can be arbitrarily smaller than the minimum among the height and width of a drawing (see Figure 1a).

Bounds for Specific Types of Drawings. For some classes of drawings (being subclasses of polyline grid drawings), we are able to prove that drawn treewidth is bounded by a sublinear function of n (the number of vertices of the graph). For example, for grid drawings – which are mappings of vertices to distinct grid points and of edges to unit-length straight lines between their endpoints (see Figure 11b in Section A.2) – we prove that the drawn treewidth (and even the *straight-line drawn treewidth*, defined ahead) is bounded by $\mathcal{O}(\sqrt{n})$. More generally, we prove that given a graph G and an orthogonal grid drawing d of G, drawn treewidth of d is $\mathcal{O}(\Delta \cdot \sqrt{\Delta \cdot \ell \cdot n} \cdot \mathsf{maxInt})$, where (i) Δ is the maximum degree in G, (ii) ℓ is the average length of the edges of G in d, and (iii) maxInt is the maximum number of edges and vertices intersected in a grid point in d.

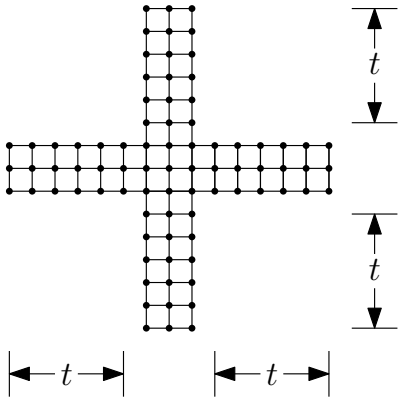
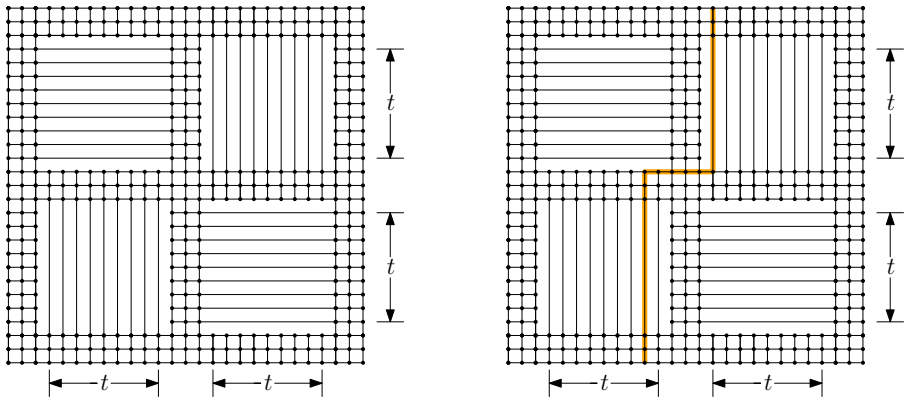


Figure 8 Example of a rectilinear drawing of a graph on n vertices with $t = \Omega(n)$. The horizontal/vertical drawn treewidth of this drawing is $\Omega(n)$.

At this point, a short discussion is in order. One of the most well-known results in graph theory about planar graphs is that every n-vertex planar graph has pathwidth (and hence treewidth) bounded by $\mathcal{O}(\sqrt{n})$ [13]. In particular, this result and generalizations thereof have found impactful applications in algorithm design, particularly of parameterized and approximation algorithms. In fact, (almost) all subexponential-time algorithms for problems on planar graphs rely on it. Here, a central component in several proofs is the planar separator theorem [3, 41, 42] (briefly mentioned earlier), which states that every n-vertex planar graph contains an $\mathcal{O}(\sqrt{n})$-sized subset of vertices (called separator) whose removal from the graph yields connected components that are each of size at most $2n/3$. Thus, due to the above-mentioned sub-quadratic bound on drawn treewidth for grid drawings, the following *conjecture* seems tempting: the drawn treewidth of any *planar* polyline grid drawing is $\mathcal{O}(\sqrt{n})$. However, we observe that the statement analogous to the planar separator theorem does not hold in our case, where our notion of a separator is that of a cutter and their sizes is, in particular, bounded from below by the size of the set of vertices associated with the cutter.

Drawbacks of Simpler Definitions for a Cutter. Lastly, we present and discuss two alternative restricted forms of cutters: *horizontal (or vertical) cutters* and *straight-line cutters*. A horizontal cutter (resp., vertical cutter) of a frame is a cutter of that frame where all vertices have the same y-coordinate (resp., x-coordinate). Then, a straight-line cutter is a cutter that is either horizontal or vertical. The replacement of cutters by horizontal/vertical cutters or straight-line cutters yields corresponding definitions of horizontal/vertical drawn tree decompositions and straight-line drawn tree decompositions, and, accordingly, of horizontal/vertical drawn treewidth and straight-line drawn treewidth. In particular, when we use these restricted forms of cutters, every frame has the shape of a rectangle. In turn, this significantly simplifies the visualization (and, possibly, also the use) of these concepts.

Unfortunately, horizontal/vertical drawn treewidth and even straight-line drawn treewidth can be arbitrarily larger than drawn treewidth. To see this, let us first consider horizontal cutters (or, symmetrically, vertical cutters), and the graph depicted in Figure 8. Notably, this graph, in fact, admits *exactly one* grid drawing (up to isomorphism) – the one depicted in the figure. Now, notice that the horizontal drawn treewidth of this drawing is $\Omega(n)$. To see



(a) (b)

Figure 9 (a) Example of a rectilinear drawing of a graph on n vertices with $t = \Omega(n)$. The straight-line drawn treewidth of this drawing is $\Omega(n)$. (b) Example of a cutter used in the drawn tree decomposition of width $O(1)$.

this, notice that, for any horizontal tree decomposition and for each of the three horizontal straight lines in the "middle" of the drawing, the rooted tree will have to contain a vertex whose associate cutter "coincides" with that line. However, the drawn treewidth of this drawing is only $\mathcal{O}(1)$, and, more generally, recall that we prove that for any grid drawing, the straight-line drawn treewidth (and hence also the drawn treewidth) is $\mathcal{O}(\sqrt{n})$. So, for example, by using only horizontal (or vertical) cutters, we will not be able to attain the subexponential-time algorithm for Grid Recognition mentioned in Section 1.3.

Nevertheless, the straight-line treewidth of the drawing in Figure 8 can be seen to be bounded by $\mathcal{O}(1)$ as well. However, regarding straight-line cutters, we consider the graph depicted in Figure 9a. Notably, every *rectilinear grid drawing* of this graph (being a generalization of a grid drawing, where edges are straight-lines of arbitrary lengths) can be obtained from the one depicted in the figure by "stretching" the drawings of some of its edges (and up to isomorphism). Now, notice that the straight-line drawn treewidth of this drawing is $\Omega(n)$. To see this, notice that every axis-parallel straight-line that intersects this graph, intersects the drawings of at least $\Omega(n)$ distinct vertices and edges of this graph. However, the drawn treewidth of this drawing is only $\mathcal{O}(1)$. To see this, consider the usage of cutters as the one depicted in Figure 9b.

1.2 Comparison with Other Graph Width Parameters

Recall that, drawn tree decomposition is based on decomposing a given polyline grid drawing of a graph. Therefore, the drawn treewidth is dependent on the polyline grid drawing of the graph. For e.g., Figures 1a and 7a depicts two different drawings of the same path which have different drawn treewidth. As path has a unique embedding, this also shows that **different drawings of the same embedded graph may have different drawn treewidth. To the best of our knowledge, our parameter is the only one that depends on the drawing (rather than the embedding or just the graph).** Thus, we compare and discuss the differences between the drawn treewidth of a given polyline drawing of the graph and some seemingly related graph width parameters, namely: treewidth, pathwidth, carving-width, dual carving-width and embedded-width. Note that, the dual carving-width and the embedded-width is only defined when the given graph is a plane graph. Specifically, we prove the following theorem.

▶ **Theorem 1.1.** *Given a graph G and a polyline drawing d of G, we have the following.*

(a) *The treewidth of G is at most 6 times the drawn treewidth of d. Moreover, the drawn treewidth of d might be arbitrary larger than the treewidth of G.*

(b) *The pathwidth of G and the drawn treewidth of d are incomparable.*

(c) *The drawn treewidth of d might be arbitrary larger than the carving-width of G.*

(d) *If G is a plane graph, the dual carving-width and the embedded-width of G might be arbitrary larger than the drawn treewidth of d.*

We now give the proof of the above theorem. Let Δ, tw, pw, and cw be the maximum degree, treewidth, pathwidth and the carving-width of G, respectively. Further, if G is a plane graph, let ℓ, dcw and emw be the maximum face size, the dual carving-width (the carving width of the dual graph), and the embedded-width of G, respectively.

Comparison with Treewidth. As mentioned earlier in Section 1.1, we prove that given a graph and a polyline drawing of it, tw is at most 6 times the drawn treewidth. Moreover, we also show that given a graph and a polyline drawing of it, the drawn treewidth of the drawing might be arbitrary larger than the treewidth of the graph (see Figure 7).

Comparison with Pathwidth. In Figure 10, we have a rectilinear grid drawing of a binary tree. By using cutters as illustrated in the figure (in orange), we can get a drawn tree decomposition of constant width. In particular, one can see that each cutter intersects a constant number of edges and vertices. Since we use only straight cutters, and the maximum degree of the graph is 3, we conclude that the width of each frame in such a drawn tree frame is bounded by a constant. Therefore, we get that the drawn treewidth of the drawing is bounded by a constant. Observe that this example can be expanded to a binary tree of any size. Furthermore, the pathwidth of a binary tree with n vertices is $\Omega(\mathsf{log}_2(n))$. So, given a graph and a polyline drawing of it, the pathwidth of the graph might be arbitrary larger than the drawn treewidth of the drawing.

On the other hand, Grid Recognition is NP-hard on graphs of pathwidth 2, and we show in this paper that the problem is XP with respect to drawn treewidth. So, given a graph and a polyline drawing of it, the drawn treewidth of the drawing might be arbitrary larger than the pathwidth of the graph. Thus, we conclude that the two parameters, pathwidth and drawn treewidth, are incomparable.

Comparison with Carving-width, Dual Carving-width and Embedded-width. It is known that $\mathsf{cw} \leq \Delta(\mathsf{tw}+1)$ [11]. As the Grid Recognition problem is NP-hard even for binary trees, we get that it is NP-hard even for graphs of carving-width at most 6. In this paper, we show that the problem is XP with respect to drawn treewidth. So, given a graph and a polyline drawing of it, the drawn treewidth of the drawing might be arbitrary larger than the carving-width of the graph.

If the given graph is plane, it is known that $\ell \leq \mathsf{dcw}$ and $\ell \leq \mathsf{emw}$ [20]. Therefore, we get that both the dual carving-width and the embedded-width of a path are at least the size of its vertex set. In this paper, we show that there exists a drawing of any path with drawn treewidth at most 16 (see Figure 1b). So, given a plane graph and a polyline drawing of it, the dual carving-width and the embedded-width of the graph might be arbitrary larger than the drawn treewidth of the drawing. Thus, we conclude that drawn treewidth differs from carving-width, dual carving-width and embedded-width.

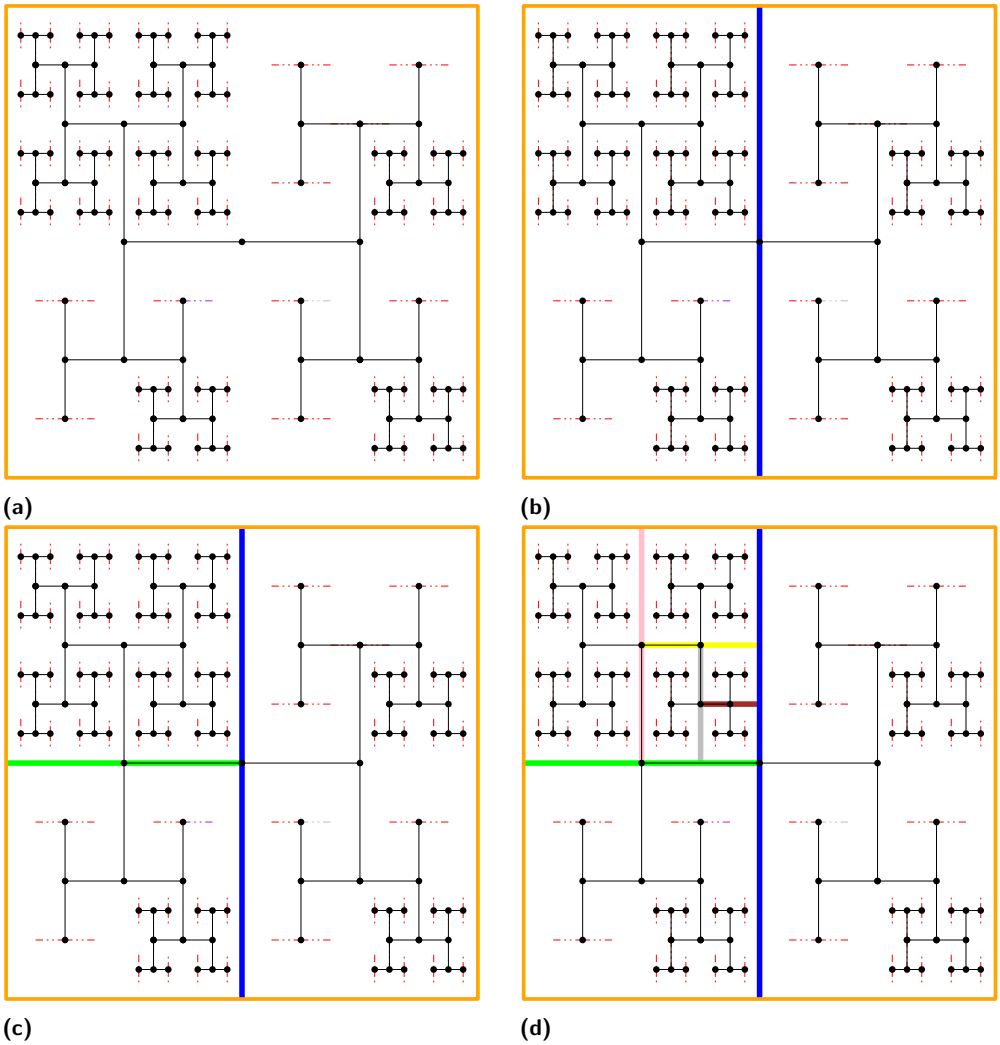


Figure 10 Example of a rectilinear drawing (in black) of a binary tree on n vertices. The rectangular is shown in orange. Examples of cutters are shown in blue, green, pink, yellow, grey and brown. Each one of them intersects $O(1)$ vertices and edges. Overall, the pathwidth of the tree is $\Omega(\log n)$, while the drawn treewidth of this drawing is $O(1)$.

1.3 Our Scheme

Here, we present (informally) our general scheme for the design of algorithms for problems in Graph Drawing parameterized by the drawn treewidth of the sought drawing (that should be, in particular, a polyline grid drawing), based on dynamic programming. For the clarity of the discussion, we first introduce the four main definitions required for the scheme and its proof of correctness. Then, we discuss the usage of our scheme – specifically, which two procedures the user should design in order to apply the scheme as a black box. Afterwards, we specify the properties that a problem should satisfy so that our scheme will solve it correctly, and the running time that will be attained. Lastly, we present some technical details concerning the scheme itself, that is, how it is executed.

Key Players: Info-Frames, Info-Cutters, Splitting and Glueing.

Info-Frames. The most basic definition required for our scheme is that of an *info-frame*. Briefly, an info-frame encodes information about the "behaviour" of the restriction of some drawing d to the interior of some particular frame f. For that purpose, the info-frame consists of five components, where the first one is, simply, the frame f. The second component is a drawing d_f that specifies the drawings of the vertices and edges (by d) of the graph on f itself. More precisely, d_f specifies which vertices of the graph are drawn on f and where are they drawn on f. Additionally, for every edge e of the graph, it specifies which are the turning points of e on f and where are these turning points drawn on f. Moreover, for the aforementioned turning points, it specifies the order in which they are encountered (when we "walk" along the drawing of e from one end to the other), and for each maximal subcurve of the drawing of e that does not contain a turning point internally, it specifies whether this subcurve is drawn on f (i.e., being a subcurve of f as well), and if yes, then it specifies the drawing of this subcurve (for which, knowing the drawings of its endpoints, we have only two options).

The third and fourth components, denoted by U_f and E_f, concern the strict interior of f. Specifically, U_f specifies which vertices of the graph are drawn strictly inside f. As for E_f, for every edge e of the graph and for each maximal subcurve of the drawing of e that does not contain a turning point internally, it specifies whether this subcurve is drawn in the strict interior of f (except for, possibly, the endpoints of the curve). We remark that the number of "sensible" choices for U_f and E_f is much smaller than it might appear to be at first glance, supposing that the graph at hand is connected. The fifth component, roughly speaking, describes the "angles" in which drawings of edges cross f using straight line segments attached to turning points. Such information is necessary, for example, to ensure that some subcurves corresponding to the drawings of the same edge lie in a single straight line, so that no bend – if forbidden by the problem at hand – occurs.

Importantly, the definition of an info-frame is independent of a specific drawing, being an "abstract" tuple of five components. Every drawing that can be described by the tuple (as discussed above) is said to be a drawing of the info-frame. So, one info-frame may describe multiple drawings, or none at all. We note that for an "abstract" five-component tuple to be an info-frame, it should satisfy various (considerably technical) properties, which, in particular, any info-frame that does describe at least one drawing must satisfy. On the one hand, these properties bound the number of possible info-frames, and, on the other hand, they are also used in the proof of correctness of our scheme.

Lastly, observe that the restriction of some drawing d to the interior of some particular frame f is *not* a drawing of a *graph*. Indeed, some edges are drawn (by d) partially in the interior of f and partially in the strict exterior of f. However, if we "enrich" the graph by placing "virtual" vertices on turning points, then the restriction of d to the interior of f will be a drawing of a graph (being a subgraph of the enriched graph). So, for technical reasons, this is exactly what we do. For this purpose, we define and work with so-called $G^\star$-drawings; however, to keep the overview short and simple, we will not discuss $G^\star$-drawings and related technical terms in this overview.

Info-Cutters. Just as we use an info-frame to encode information about the "behaviour" of a drawing d with respect to a frame f, we use an *info-cutter* of an info-frame to encode information about the "behaviour" of d with respect to a cutter c of f. Rather than directly describing how d is drawn on c and how d is "split" by c inside f, we find it easier to indirectly describe this information by defining an info-cutter based on two info-frames corresponding to the frames obtained by cutting f with c (later, for the dynamic programming implementation,

we can thus immediately know to which already computed entries to refer). Observe that, in particular, the two frames being part of these two info-frames contain c, and, thus, these two info-frames capture the aforementioned information.

To be more precise, an info-cutter C of an info-frame F, where the first component of F is some frame f, is a triple (c, F_1, F_2), where, in particular, c is a cutter of f, and F_1 and F_2 are info-frames for the two frames obtained by cutting f with c. Additionally, for such a triple to be an info-cutter, it should satisfy (considerably technical) properties, which, in particular, any info-cutter that does describe at least one drawing must satisfy. Very briefly, these properties validate consistency between the information described by F, F_1 and F_2. This is more complicated than it might appear to be at first glance, since, even on the cutter c, F_1 and F_2 might describe the existence of different virtual vertices (having different turning points). For the sake of simplicity, we do not discuss these details in the overview.

Splitting and Glueing. First, let us consider the *splitter function*, which, for our scheme, is used only for the proof of correctness (where its input is assumed to contain a subdrawing of a hypothetical solution drawing). Given an info-frame F whose first component (being a frame) is f, a drawing d restricted to the interior of f that is compatible with the description encoded by F, and a cutter c of f, the splitter function returns an info-cutter $C = (c, F_1, F_2)$ and two drawings, d_1 and d_2. Let f_1 (f_2) be the first component of F_1 (F_2). Briefly, we define the output such that d_1 and d_2 would be the subdrawings of d restricted to the interiors of f_1 and f_2, respectively, and F_1 and F_2 would be the info-frames that describe d_1 and d_2, respectively.

The *glue function* is, intuitively, the "inverse" of the split function, and it is used algorithmically in our scheme. Its input consists of an info-frame F, an info-cutter $C = (c, F_1, F_2)$ of F, a drawing d_1 of F_1 and a drawing d_2 of F_2. Roughly speaking, this function aims to "glue" d_1 and d_2 into a single drawing d that is restricted to the interior of f, being the first component of F, and that should be compatible with the description encoded by F; of course, this operation might be impossible, and then the function simply announces that. Among other proofs concerning these functions, we show, in particular, that the specific way in which we define the splitter and glue functions (not described in the overview) ensures that, if we apply the glue function on an output of the splitter function, we are able to reconstruct the drawing given as input to the splitter function.

The User's Point of View. For the execution of the scheme, we expect the user to provide four components: some universe denoted by INF, and three algorithmic procedures (that will be defined immediately). All of these components are problem-dependent.

- The first procedure, termed *classifier* and denoted by $\mathsf{Classifer}$, is given an info-frame F and a corresponding drawing d, and it returns an element from INF. Intuitively, this element describes the equivalence class of d. So, we say that two drawings corresponding to the same info-frame are *equivalent* if the classifier associates them with the same element.
- The second procedure, termed *classifier algorithm*, is given an info-frame F, an info-cutter $C = (c, F_1, F_2)$ of F and $I_1, I_2 \in \mathsf{INF}$, and it returns $I' \in \mathsf{INF}$ such that: For any two drawings d_1 and d_2 corresponding to F_1 and F_2, respectively, such that $\mathsf{Classifier}(F_1, d_1) = I_1$ and $\mathsf{Classifier}(F_2, d_2) = I_2$, we have $\mathsf{Classifier}(F, d) = I'$ where $d = \mathsf{Glue}(F, C, d_1, d_2)$. In particular, notice that any two drawings of the same two equivalence classes always yield (when being glued) a drawing of the same equivalence class – this justifies our usage of the term *equivalence* in this context.

- The third procedure, termed *leaf solver*, is given an info-frame F whose frame does not contain any grid point in its strict interior, and for every $I' \in \mathsf{INF}$, it returns "yes" if and only if there exists a drawing d corresponding to F such that $\mathsf{Classifier}(F, d) = I'$. Practically, we require this procedure to solve the basis of our dynamic programming computation, corresponding to info-frames whose frames do not contain any grid point in their strict interiors.

The scheme, once given these components, can be executed in a black box fashion. For the sake of simplicity of the overview, we do not discuss the technical details of the execution itself (as a white box) here.

To Which Type of Problems Does Our Scheme Apply? Roughly speaking, we prove that our scheme can be applied to any graph drawing problem Π such that:

1. Every instance of Π contains, in particular, a connected graph G, dimensions h and w for the sought drawing (which are, usually, bounded from above by the number of vertices n of G), and the parameter k (being any non-negative integer).
2. The objective is to determine whether G admits a polyline grid drawing bounded by rectangle of dimensions $h \times w$, whose drawn treewidth is at most k, and that satisfies various problem-specific properties (for some examples, see Section 1.4).
3. The user can design the three algorithmic procedures discussed above.

For any such problem Π, we prove that the runtime of the scheme is bounded by

$$\mathcal{O}(k \cdot h \cdot w \cdot n)^{\mathcal{O}(k)} \cdot |\mathsf{INF}|^{\mathcal{O}(1)} \cdot \left(2^{\mathcal{O}(\Delta \cdot k)} \cdot \mathsf{T2} + \mathsf{T3}\right),$$

where $\mathsf{T2}$ and $\mathsf{T3}$ bound the runtimes of the second and third procedures provided by the user, and Δ is the maximum degree of G. In particular, if $h, w, |\mathsf{INF}|, \mathsf{T2}$ and $\mathsf{T3}$ can be bounded by $n^{\mathcal{O}(1)}$ (which is the case for many applications, such as grid recognition and orthogonal compaction), then the runtime above simplifies to $n^{\mathcal{O}(k)}$, that is, we obtain an XP-algorithm.

1.4 Applications of Our Scheme to Problems in Graph Drawing

For most of the problems considered in this paper, the time complexity of our scheme can be bounded by $n^{\mathcal{O}(k)}$, where k is the input parameter that upper bounds the drawn treewidth of the output drawing. We remark that the formal definitions of these problems are relegated to Section A.3.

Grid Recognition. We first consider the relatively simple Grid Recognition problem in order to demonstrate the application of our scheme. Here, given a (connected) graph G, the objective is to determine whether G is a grid graph, that is, whether it admits a grid drawing. The Grid Recognition problem was first proved to be NP-hard in 1987, on ternary trees of pathwidth 3 [7]. Two years later in 1989, the problem was proved to be NP-hard even on binary trees [31]. Recently in 2021, the problem was proved to be NP-hard even on trees of pathwidth 2 [34]. In the same paper, it was also proved that the problem is polynomial time solvable on graphs of pathwidth 1. A year later in 2022, it was proved that even if we require all the internal faces of the drawing to be rectangles, the problem is still NP-hard even for biconnected graphs [2]. In the same paper, it was also proved that if we require all the faces of the drawing to be rectangles (including outer face), the problem is cubic time solvable.

As we deal with the parameterized version of this problem where the parameter is the drawn treewidth of the sought drawing (or, more precisely, an upper bound on it), we are also given k as input. We prove the following result.

▶ **Theorem 1.2.** *There exists an algorithm that solves the* Grid Recognition *problem in time* $n^{\mathcal{O}(k)}$.

Since for grid drawings, we also prove that $k \leq \mathcal{O}(\sqrt{n})$, we get the following corollary.

▶ **Corollary 1.3.** *There exists an algorithm that solves the* Grid Recognition *problem in time* $n^{\mathcal{O}(\sqrt{n})}$.

Thus, we obtain a subexponential-time algorithm for Grid Recognition, matching the running time of the current best known algorithm for this problem [21].

Crossing and Bend Minimization. For our second application, we study a variant of the Crossing Minimization problem. The Crossing Minimization problem is one of the most fundamental graph layout problems. It was shown to be NP-complete by Garey and Johnson in 1983 [29]. Later, it was proved to be NP-complete even on graph of maximum degree 3 [36] and also on *almost planar graphs* which are graphs that can be made planar by removing a single edge [15]. It was also shown that the problem remains NP-hard even if the cyclic order of the neighbours around each vertex is fixed and to be respected by the resulting drawing [44]. On the positive side, it is known the problem is FPT with respect to the number of crossings [32, 38] and also with respect to the vertex cover [37]. There are many other variants of this problem which are studied in the literature. One of them concerns with minimizing the number of pairwise crossing edges in any straight-line drawing of the graph. This problem is known to be NP-hard [12] (and even $\exists\mathbb{R}$-complete [46]). For more information about the crossing minimization and its variants, we refer to the survey [48].

A related problem is the Bend Minimization problem. Given a graph G, the Bend Minimization problem asks for an orthogonal grid drawing of G with minimum number of total bends. The problem was proved to be NP-complete in 2001, even when there are no bends [30]. On the positive side, if the input graph is plane, the problem can be solved in polynomial time [47]. When the input graph is not planar, there are polynomial time algorithms for subclasses of planar graphs, namely planar graphs with maximum degree 3 [17, 6, 25, 45] and series-parallel graphs [49].

We study the Straight-line Grid Crossing Minimization problem where the sought drawing should be a straight-line grid drawing. Here, given a (connected) graph G and $h, w \in \mathbb{N}$, the objective is to determine a straight-line grid drawing of G bounded by a rectangle of dimension $h \times w$ with minimum number of crossings, if one exists. Similar to the previous example, as we study the parameterized version of this problem, we are also given k as input. We prove the following result.

▶ **Theorem 1.4.** *There exists an algorithm that solves* Straight-line Grid Crossing Minimization *problem in time* $\mathcal{O}((k \cdot h \cdot w \cdot n)^{\mathcal{O}(k)} \cdot 2^{\mathcal{O}(\Delta \cdot k)})$*, where* Δ *is the maximum degree of the input graph.*

More generally, our scheme can be applied to a very wide class of problems of such flavor; in particular, every problem where:

- The input consists of (some or all of) the following: a graph G; $\mathsf{cross} : E(G) \to \mathbb{N}_0 \cup \infty$; $\mathsf{bend} : E(G) \to \mathbb{N}_0 \cup \infty$, and $C, B, k \in \mathbb{N}_0 \cup \{\infty\}$. Here, $E(G)$ is the edge set of G, and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.

- The objective is to determine whether G admits a drawing that is *(i)* a grid drawing, or *(ii)* a rectilinear drawing, or *(iii)* an orthogonal grid drawing, or *(iv)* a straight-line grid drawing, or *(v)* a polyline grid drawing, such that:
  - For every edge $e \in E(G)$, the drawing of e has at most $\mathsf{cross}(e)$ crossings and at most $\mathsf{bend}(e)$ bends.
  - In total, we have at most C crossings and at most B bends.

Further, the scheme can be applied to various variants of the above generic problem that were studied in the literature. For example, we can specify, for every edge, whether it should be crossed an even or odd number of times. Similarly, we can also consider the weighted crossing number.

Orthogonal Compaction. Lastly, we note that our scheme can also be applied to problems of flavors quite different than the above. As an example, we consider the Orthogonal Compaction problem. Here, given a planar orthogonal representation H of a connected planar graph G, the objective is to compute a minimum-area drawing of H. The Orthogonal Compaction problem was first proved to be NP-hard on general graphs in 2001 [43]. Later, it was shown that the problem is NP-hard even on cycles [26], ruling out an FPT algorithm with respect to treewidth, unless P=NP. On the positive side, it was proved that the problem is linear time solvable for a restricted class of planar orthogonal representation [14]. Recently, it was also shown that the problem is FPT with respect to number of "kitty corner vertices", a parameter central to the problem [23].

Similar to the previous examples, as we study the parameterized version of this problem, we are also given k as input. We prove the following result.

▶ **Theorem 1.5.** *There exists an algorithm that solves the* Orthogonal Compaction *problem in time* $n^{\mathcal{O}(k)}$.

References

1. Hugo A. Akitaya, Maarten Löffler, and Irene Parada. How to fit a tree in a box. *Graphs Comb.*, 38(5):155, 2022. doi:10.1007/s00373-022-02558-z.
2. Carlos Alegría, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Fabrizio Grosso, and Maurizio Patrignani. Unit-length rectangular drawings of graphs. In Patrizio Angelini and Reinhard von Hanxleden, editors, *Graph Drawing and Network Visualization - 30th International Symposium, GD 2022, Tokyo, Japan, September 13-16, 2022, Revised Selected Papers*, volume 13764 of *Lecture Notes in Computer Science*, pages 127–143. Springer, 2022. doi:10.1007/978-3-031-22203-0_10.
3. Noga Alon, Paul D. Seymour, and Robin Thomas. Planar separators. *SIAM J. Discret. Math.*, 7(2):184–193, 1994. doi:10.1137/S0895480191198768.
4. Michael J. Bannister, Sergio Cabello, and David Eppstein. Parameterized complexity of 1-planarity. *Journal of Graph Algorithms and Applications*, 22(1):23–49, 2018.
5. Michael J. Bannister and David Eppstein. Crossing minimization for 1-page and 2-page drawings of graphs with bounded treewidth. *Journal of Graph Algorithms and Applications*, 22(4):577–606, 2018.
6. Giuseppe Di Battista, Giuseppe Liotta, and Francesco Vargiu. Spirality and optimal orthogonal drawings. *SIAM J. Comput.*, 27(6):1764–1811, 1998. doi:10.1137/S0097539794262847.
7. 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.
8. Sujoy Bhore, Robert Ganian, Fabrizio Montecchiani, and Martin Nöllenburg. Parameterized algorithms for book embedding problems. *Journal of Graph Algorithms and Applications*, 24(4):603–620, 2020.

9 Therese Biedl. On area-optimal planar graph drawings. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 198–210. Springer, 2014. doi:10.1007/978-3-662-43948-7_17.

10 Therese Biedl and Debajyoti Mondal. On upward drawings of trees on a given grid. In Fabrizio Frati and Kwan-Liu Ma, editors, *Proc. 25th International Symposium on Graph Drawing and Network Visualization (GD)*, volume 10692 of *LNCS*, pages 318–325. Springer, 2017. doi:10.1007/978-3-319-73915-1_25.

11 Therese Biedl and Martin Vatshelle. The point-set embeddability problem for plane graphs. *Int. J. Comput. Geom. Appl.*, 23(4-5):357–396, 2013. doi:10.1142/S0218195913600091.

12 Daniel Bienstock. Some provably hard crossing number problems. *Discrete & Computational Geometry*, 6(3):443–459, 1991.

13 Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.

14 Stina S Bridgeman, Giuseppe Di Battista, Walter Didimo, Giuseppe Liotta, Roberto Tamassia, and Luca Vismara. Turn-regularity and optimal area drawings of orthogonal representations. *Computational Geometry*, 16(1):53–93, 2000.

15 Sergio Cabello and Bojan Mohar. Adding one edge to planar graphs makes crossing number and 1-planarity hard. *SIAM Journal on Computing*, 42(5):1803–1829, 2013.

16 Hubert Chan. A parameterized algorithm for upward planarity testing. In *European Symposium on Algorithms, ESA*, pages 157–168. Springer, 2004.

17 Yi-Jun Chang and Hsu-Chun Yen. On bend-minimized orthogonal drawings of planar 3-graphs. In Boris Aronov and Matthew J. Katz, editors, *33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017, Brisbane, Australia*, volume 77 of *LIPIcs*, pages 29:1–29:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.SoCG.2017.29.

18 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

19 Giordano Da Lozzo, David Eppstein, Michael T. Goodrich, and Siddharth Gupta. Subexponential-time and FPT algorithms for embedded flat clustered planarity. In *Graph-Theoretic Concepts in Computer Science - 44th International Workshop, WG 2018, Cottbus, Germany, June 27-29, 2018, Proceedings*, pages 111–124, 2018. doi:10.1007/978-3-030-00256-5_10.

20 Giordano Da Lozzo, David Eppstein, Michael T. Goodrich, and Siddharth Gupta. C-planarity testing of embedded clustered graphs with bounded dual carving-width. *Algorithmica*, 83(8):2471–2502, 2021. doi:10.1007/s00453-021-00839-2.

21 Peter Damaschke. Enumerating grid layouts of graphs. *J. Graph Algorithms Appl.*, 24(3):433–460, 2020.

22 Emilio Di Giacomo, Giuseppe Liotta, and Fabrizio Montecchiani. Orthogonal planarity testing of bounded treewidth graphs. *Journal of Computer and System Sciences*, 125:129–148, 2022. doi:10.1016/j.jcss.2021.11.004.

23 Walter Didimo, Siddharth Gupta, Philipp Kindermann, Giuseppe Liotta, Alexander Wolff, and Meirav Zehavi. Parameterized approaches to orthogonal compaction. In Leszek Gasieniec, editor, *SOFSEM 2023: Theory and Practice of Computer Science - 48th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2023, Nový Smokovec, Slovakia, January 15-18, 2023, Proceedings*, volume 13878 of *Lecture Notes in Computer Science*, pages 111–125. Springer, 2023. doi:10.1007/978-3-031-23101-8_8.

24 Walter Didimo and Giuseppe Liotta. Computing orthogonal drawings in a variable embedding setting. In *Proceedings of the 9th International Symposium on Algorithms and Computation, ISAAC*, pages 80–89. Springer, 1998.

25 Walter Didimo, Giuseppe Liotta, and Maurizio Patrignani. Bend-minimum orthogonal drawings in quadratic time. In Therese Biedl and Andreas Kerren, editors, *Graph Drawing and Network Visualization - 26th International Symposium, GD 2018, Barcelona, Spain, September 26-28, 2018, Proceedings*, volume 11282 of *Lecture Notes in Computer Science*, pages 481–494. Springer, 2018. doi:10.1007/978-3-030-04414-5_34.

26 William S. Evans, Krzysztof Fleszar, Philipp Kindermann, Noushin Saeedi, Chan-Su Shin, and Alexander Wolff. Minimum rectilinear polygons for given angle sequences. *Comput. Geom.*, 100:101820, 2022. doi:10.1016/j.comgeo.2021.101820.

27 Mike Fellows, Panos Giannopoulos, Christian Knauer, Christophe Paul, Frances A. Rosamond, Sue Whitesides, and Nathan Yu. Milling a graph with turn costs: A parameterized complexity perspective. In *Proceedings of the 36th International Workshop on Graph Theoretic Concepts in Computer Science, WG*, pages 123–134, 2010.

28 Robert Ganian, Fabrizio Montecchiani, Martin Nöllenburg, and Meirav Zehavi. Parameterized complexity in graph drawing (dagstuhl seminar 21293). *Dagstuhl Reports*, 11(6):82–123, 2021.

29 Michael R Garey and David S Johnson. Crossing number is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3):312–316, 1983.

30 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.

31 Angelo Gregori. Unit-length embedding of binary trees on a square grid. *Information Processing Letters*, 31(4):167–173, 1989.

32 Martin Grohe. Computing crossing numbers in quadratic time. *Journal of Computer and System Sciences*, 68(2):285–302, 2004.

33 Siddharth Gupta, Guy Sa'ar, and Meirav Zehavi. Drawn tree decomposition: New approach for graph drawing problems, 2023. arXiv:2310.05471.

34 Siddharth Gupta, Guy Sa'ar, and Meirav Zehavi. Grid recognition: Classical and parameterized computational perspectives. *Journal of Computer and System Sciences*, 136:17–62, 2023. doi:10.1016/j.jcss.2023.02.008.

35 Patrick Healy and Karol Lynch. Two fixed-parameter tractable algorithms for testing upward planarity. *International Journal of Foundations of Computer Science*, 17(05):1095–1114, 2006.

36 Petr Hliněný. Crossing number is hard for cubic graphs. *Journal of Combinatorial Theory, Series B*, 96(4):455–471, 2006.

37 Petr Hliněný and Abhisekh Sankaran. Exact crossing number parameterized by vertex cover. In *Proceedings of the 27th International Symposium on Graph Drawing and Network Visualization, GD*, pages 307–319, 2019.

38 Ken-ichi Kawarabayashi and Buce Reed. Computing crossing number in linear time. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, STOC*, pages 382–390, 2007.

39 Marcus Krug and Dorothea Wagner. Minimizing the area for planar straight-line grid drawings. In Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors, *Graph Drawing, 15th International Symposium, GD 2007, Sydney, Australia, September 24-26, 2007. Revised Papers*, volume 4875 of *Lecture Notes in Computer Science*, pages 207–212. Springer, 2007. doi:10.1007/978-3-540-77537-9_21.

40 Giuseppe Liotta, Ignaz Rutter, and Alessandra Tappini. Parameterized complexity of graph planarity with restricted cyclic orders. *J. Comput. Syst. Sci.*, 135:125–144, 2023. doi:10.1016/j.jcss.2023.02.007.

41 Richard J Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

42 Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *J. Comput. Syst. Sci.*, 32(3):265–279, 1986. doi:10.1016/0022-0000(86)90030-9.

43 Maurizio Patrignani. On the complexity of orthogonal compaction. *Computational Geometry*, 19(1):47–67, 2001.

44 Michael J. Pelsmajer, Marcus Schaefer, and Daniel Stefankovic. Crossing numbers of graphs with rotation systems. *Algorithmica*, 60(3):679–702, 2011.

45 Md. Saidur Rahman, Noritsugu Egi, and Takao Nishizeki. No-bend orthogonal drawings of subdivisions of planar triconnected cubic graphs. *IEICE Trans. Inf. Syst.*, 88-D(1):23–30, 2005. URL: `http://search.ieice.org/bin/summary.php?id=e88-d_1_23&category=D&year=2005&lang=E&abst=`.

46 Marcus Schaefer. Complexity of some geometric and topological problems. In *Proceedings of the 18th International Symposium on Graph Drawing and Network Visualization, GD*, pages 334–344. Springer, 2009.

47 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`.

48 Meirav Zehavi. Parameterized analysis and crossing minimization problems. *Computer Science Review*, 45:100490, 2022. `doi:10.1016/j.cosrev.2022.100490`.

49 Xiao Zhou and Takao Nishizeki. Orthogonal drawings of series-parallel graphs with minimum bends. *SIAM J. Discret. Math.*, 22(4):1570–1604, 2008. `doi:10.1137/060667621`.

A Preliminaries

In this paper, we only consider finite simple undirected graphs, unless stated otherwise. Moreover, we refer to straight line segments as line segments, unless stated otherwise. Let $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. For $k, i, j \in \mathbb{N}$, we denote $[k] = \{1, 2, \ldots k\}$ and $[i, j] = \{i, i+1, \ldots, j\}$.

A.1 Graph Notation and Decompositions

For a graph $G = (V, E)$ and a subset of vertices $U \subseteq V$, we denote by $G[U]$ the subgraph of G induced by U. For a given subset $V' \subseteq V$ of vertices, we define the *boundary of* V' as the set of vertices in V' that are adjacent to a vertex in $V \setminus V'$:

▶ **Definition A.1 (Boundary).** *Let $G = (V, E)$ be a graph. Let $V' \subseteq V$. Then the* boundary *of V' in G, denoted by $B_G(V')$, is the set of vertices of V' that have a neighbor in $V \setminus V'$, i.e., $B_G(V') = \{v' \in V' \mid$ there exists $v \in V \setminus V'$ such that $\{v, v'\} \in E\}$.*

When the graph G is clear from the context, we drop it from the subscript. Given a path P, we represent P as a sequence of vertices $v_1, v_2, \ldots, v_k$, such that $\{v_i, v_{i+1}\}$ is an edge in P for every $1 \leq i \leq k-1$. Similarly, given a cycle C, we represent C as a sequence of vertices $v_1, v_2, \ldots, v_k$, such that $v_1 = v_k$ and $\{v_i, v_{i+1}\}$ is an edge in C for every $1 \leq i \leq k-1$. Note that we use the terms path and cycle to refer to simple path and cycles. We now define the concepts of a *tree decomposition* and a *path decomposition*.

▶ **Definition A.2 (Tree Decomposition).** *A* tree decomposition *of a graph $G = (V, E)$ is a pair $(\mathcal{T} = (V_T, E_T), \beta : V_T \to 2^V)$ where $\mathcal{T}$ is a tree such that:*

1. *For every $v \in V$, the subgraph of $\mathcal{T}$ induced by $\{x \in V_T \mid v \in \beta(x)\}$ is non-empty and connected.*
2. *For every $\{u, v\} \in E$, there exists $x \in V_T$ such that $\{u, v\} \subseteq \beta(x)$.*

The width *of $(\mathcal{T}, \beta)$ is defined to be $\max_{x \in V_T} |\beta(x)| - 1$. For every $x \in V_T$, $\beta(x)$ is called a* bag. *The* treewidth *of a graph G is the minimum width of any tree decomposition of G.*

▶ **Definition A.3 (Path Decomposition).** *A* path decomposition *of a graph $G = (V, E)$ is a pair $(\mathcal{P} = (V_P, E_P), \beta : V_P \to 2^V)$ where $\mathcal{P}$ is a path such that:*

1. *For every $v \in V$, the subgraph of $\mathcal{P}$ induced by $\{x \in V_P \mid v \in \beta(x)\}$ is non-empty and connected.*
2. *For every $\{u, v\} \in E$, there exists $x \in V_P$ such that $\{u, v\} \subseteq \beta(x)$.*

The width *of $(\mathcal{P}, \beta)$ is defined to be $\max_{x \in V_P} |\beta(x)| - 1$. For every $x \in V_P$, $\beta(x)$ is called a* bag. *The* pathwidth *of a graph G is the minimum width of any path decomposition of G.*

A.2 Graph Drawing

For a given graph G, a *drawing of* G on the plane is a mapping of the vertices to distinct points of $\mathbb{R}^2$ and of the edges to simple curves in $\mathbb{R}^2$, connecting the images of their endpoints. A drawing of a graph is *planar* if no pair of edges, or an edge and a vertex, cross except at a common endpoint. Two planar drawings of the same graph are *equivalent* if they determine the same *rotation* at each vertex, that is, the same circular ordering for the edges around each vertex. An *embedding* is an equivalence class of planar drawings.

Given a drawing d of G, we represent d as a pair of functions (d_V, d_E) as follows. The function $d_V : V \to \mathbb{R} \times \mathbb{R}$ is an injection, which maps each vertex v of G to a point (i, j) in the plane; then, i and j are also denoted as $d_{\mathsf{x}}(v)$ and $d_{\mathsf{y}}(v)$, respectively, that is, $d_V(v) = (d_{\mathsf{x}}(v), d_{\mathsf{y}}(v))$. The function $d_E : E \to \mathcal{C}$, where $\mathcal{C}$ is the set of all simple curves in the plane, maps each edge $\{u, v\} \in E$ to a simple curve $c \in \mathcal{C}$ between $d_V(u)$ and $d_V(v)$. For simplicity, we refer to (d_V, d_E) as one function, $d : V \cup E \to \{\mathbb{R} \times \mathbb{R}\} \cup \mathcal{C}$, such that $d(v) = d_V(v)$ for every $v \in V$, and $d(\{u, v\}) = d_E(\{u, v\})$ for every $\{u, v\} \in E$. We call V and E the *vertex set* and the *edge set associated with* d, respectively. Let d be a drawing of a graph G, and let $p \in \mathbb{R}^2$ be a point. We say that p *is on* d if p is on the image of an edge of G in d or p is the image of a vertex of G in d. We denote by $\mathsf{PlanePoints}(d)$ the set of points on d.

For two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ in the plane, we denote the line segment joining the points by $\ell(p_1, p_2)$. For four points, $p_i = (x_i, y_i) \in \mathbb{R}^2$ for every $1 \leq i \leq 4$, we say that $\ell(p_1, p_2)$ *crosses* $\ell(p_3, p_4)$ if the line segments $\ell(p_1, p_2)$ and $\ell(p_3, p_4)$ cross except at $p_i = (x_i, y_i)$ for every $1 \leq i \leq 4$. Let a and b be two points in $\mathbb{R}^2$ and let $\epsilon > 0$. We denote $\ell(a, a_\epsilon)$ by $\mathsf{line}_\epsilon(a, b)$, where a_ϵ is the point on the line $\ell(a, b)$ at distance ϵ from a if it exists. For a pair of points (p_1, p_2), and a point p', where $p_1, p_2, p' \in \mathbb{R}^2$, we say that $\ell(p_1, p_2)$ *intersects* p' if p' is on the line $\ell(p_1, p_2)$, including its endpoints. We use the term *grid points* to refer to the infinite set of points $(x, y) \in \mathbb{R}^2$ where $x, y \in \mathbb{N}_0$. Given two distinct grid points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, we say that $p_1 < p_2$ if $x_1 < x_2$ or $x_1 = x_2$ and $y_1 < y_2$.

A *drawn graph* is a graph with a prescribed drawing. A *plane graph* is a drawn graph whose prescribed drawing is planar. A drawing of a graph is called a *straight-line* drawing if the edges are mapped to line segments, connecting the images of their endpoints. We define a *straight-line path (cycle)* as a plane path (cycle), where the vertices are mapped to grid points and edges are mapped to line segments connecting the images of their endpoints. We denote by $\mathcal{P} \subset \mathcal{C}$ the (infinite) set of straight-line paths in $\mathbb{R}^2$. Moreover, we alternatively denote any path $P = (v_1, \ldots, v_k) \in \mathcal{P}$ by the sequence $(p_1, \ldots, p_k)$, where $p_i \in \mathbb{R}^2$ is the image of the vertex v_i in P, for every $1 \leq i \leq k$. We define an *axis-parallel path (cycle)* as a straight-line path (cycle), where every edge of the path is parallel to the X- or Y- axis. For an axis-parallel path $P = (p_1, \ldots, p_k)$, we denote by $|P|$ the (Euclidean) *length* of P, that is, $|P| = |p_2 - p_1| + |p_3 - p_2| + \ldots + |p_k - p_{k-1}|$. Next, we define a *grid drawing* of a graph G as a straight-line drawing of G where the vertices are mapped to grid points and the edges are mapped to (axis-parallel) unit length line segments (e.g., see Figure 11b):

▶ **Definition A.4 (Straight-Line Grid Drawing).** *Let G be a graph. A* straight-line grid *drawing d of G is a straight-line drawing d of G such that (i) for every $u \in V$, $d(u)$ is a grid point (ii) For every $\{u, v\}, \{u', v'\} \in E$, $d(\{u, v\})$ and $d(\{u', v'\})$ are intersected in at most one point.*

▶ **Definition A.5 (Grid Drawing).** *Let $G = (V, E)$ be a graph. A* grid drawing *d of G is a drawing $d : V \cup E \to \mathbb{N}_0 \times \mathbb{N}_0 \cup \mathcal{P}$ such that if $\{u, v\} \in E$ then $|d_{\mathsf{x}}(u) - d_{\mathsf{x}}(v)| + |d_{\mathsf{y}}(u) - d_{\mathsf{y}}(v)| = 1$.*

We now extend the concept of a grid drawing to a *rectilinear grid drawing*, where the edges are mapped to variable length line segments parallel to the axes (e.g., see Figure 11c):

▶ **Definition A.6 (Rectilinear Grid Drawing).** *Let $G = (V, E)$ be a graph. A* rectilinear grid drawing *d of G is a drawing $d : V \cup E \to \mathbb{N}_0 \times \mathbb{N}_0 \cup \mathcal{P}$ of G, such that for every edge $\{u, v\} \in E$, $d(\{u, v\})$ is a line segment between $d(u)$ and $d(v)$ such that $d_{\mathsf{x}}(u) = d_{\mathsf{x}}(v)$ or $d_{\mathsf{y}}(u) = d_{\mathsf{y}}(v)$.*

Further, we extend the concept of a rectilinear grid drawing to an *orthogonal grid drawing*, where the edges are mapped to straight-line paths, such that the edges of these paths are mapped to line segments parallel to the axes (e.g., see Figure 11d):

▶ **Definition A.7 (Orthogonal Grid Drawing).** *Let $G = (V, E)$ be a graph. An* orthogonal grid drawing *d of G is a drawing $d : V \cup E \to \mathbb{N}_0 \times \mathbb{N}_0 \cup \mathcal{P}$ of G, such that for every edge $\{u, v\} \in E$, $d(\{u, v\})$ is an axis-parallel path between $d(u)$ and $d(v)$.*

Finally, we extend the concept of an orthogonal grid drawing to a *polyline grid drawing*, where the edges are mapped to straight-line paths instead of axis-parallel paths (e.g., see Figure 11e).

▶ **Definition A.8 (Polyline Grid Drawing).** *Let $G = (V, E)$ be a graph. A* polyline grid drawing *d of G is a drawing $d : V \cup E \to \mathbb{N}_0 \times \mathbb{N}_0 \cup \mathcal{P}$ of G.*

A.3 Problem Definitions

In this subsection, we give the definitions for the problems we will solve using our new concept.

▶ **Definition A.9 (Grid Recognition Problem).** *The* Grid Recognition *problem is, given a graph G, to determine whether G has a grid drawing.*

▶ **Definition A.10 (Crossing Minimization Problem on Straight-Line Grid Drawings).** *The* Straight-line Grid Crossing Minimization *problem is, given a graph G and $h, w \in \mathbb{N}$, to construct a straight-line grid drawing d of G (if one exists) such that: (i) d is strictly bounded by $R_{h,w}$,(ii) d has minimum number of crossings out of all the straight-line grid drawings of G which are strictly bounded by $R_{h,w}$. If such a drawing does not exists, return "no-instance".*

In the Orthogonal Compaction problem we get a connected graph G. We assume to have an order on the vertices, that is, for every $u, v \in V$ such that $u \neq v$, either $u > v$ or $v < u$. In addition to G, we have, for every $\{u, v\} \in E$ where $u > v$, the relative position of v compered to u, that is, the *direction* of the $\{u, v\}$ from u to v. We denote these directions by $\mathsf{U}, \mathsf{D}, \mathsf{L}$ and R; this stands for "up", "down", "left" and "right", respectively. We assume that there exists a planar rectilinear grid drawing of G such that for every $\{u, v\} \in E$, the relative position of v compered to u is as given as input. Our goal is to find such a drawing of minimum area. We start by defining the problem formally. For this purpose, we first have the following definition:

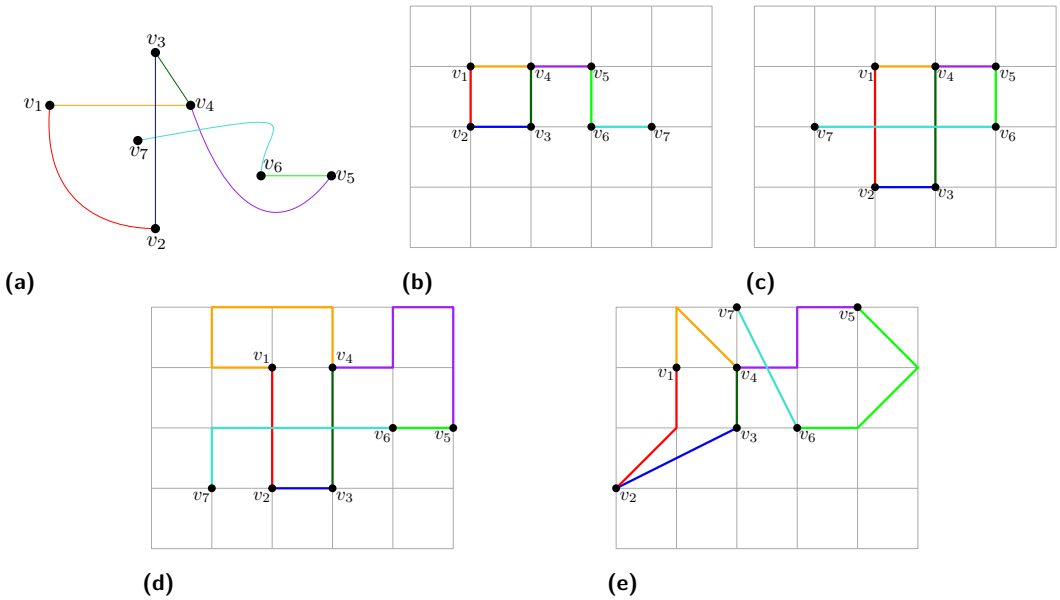


Figure 11 Different drawings (defined in Definitions A.5-A.8) of the graph G shown in (a). A grid, a rectilinear grid, an orthogonal grid and a polyline grid drawings of G are shown in (b), (c), (d) and (e), respectively.

▶ **Definition A.11 (Drawing Respects an Edge Direction).** *Let G be a connected graph, let $\{u, v\} \in E$ such that $u > v$, and let $\mathsf{dir}_{\{u,v\}} \in \{\mathsf{U}, \mathsf{D}, \mathsf{L}, \mathsf{R}\}$. Let d be a rectilinear grid drawing of G. We say that d* respects *$\mathsf{dir}_{\{u,v\}}$ if the following conditions are satisfied*

1. *If $\mathsf{dir}_{\{u,v\}} = \mathsf{U}$, then $d_{\mathsf{x}}(v) = d_{\mathsf{x}}(u)$ and $d_{\mathsf{y}}(v) > d_{\mathsf{y}}(u)$.*
2. *If $\mathsf{dir}_{\{u,v\}} = \mathsf{D}$, then $d_{\mathsf{x}}(v) = d_{\mathsf{x}}(u)$ and $d_{\mathsf{y}}(v) < d_{\mathsf{y}}(u)$.*
3. *If $\mathsf{dir}_{\{u,v\}} = \mathsf{L}$, then $d_{\mathsf{y}}(v) = d_{\mathsf{y}}(u)$ and $d_{\mathsf{x}}(v) < d_{\mathsf{x}}(u)$.*
4. *If $\mathsf{dir}_{\{u,v\}} = \mathsf{R}$, then $d_{\mathsf{y}}(v) = d_{\mathsf{y}}(u)$ and $d_{\mathsf{x}}(v) > d_{\mathsf{x}}(u)$.*

Now, we define the problem Orthogonal Compaction as follows:

▶ **Definition A.12 (Orthogonal Compaction Problem).** *Let G be a connected graph. For every $\{u, v\} \in E$ let $\mathsf{dir}_{\{u,v\}} \in \{\mathsf{U}, \mathsf{D}, \mathsf{L}, \mathsf{R}\}$. The* Orthogonal Compaction *problem is to find a planar rectilinear grid drawing d of G such that (i) for every $\{u, v\} \in E$, d respects $\mathsf{dir}_{\{u,v\}}$, and (ii) d is strictly bounded by $R_{h,w}$ such that $(h-1) \cdot (w-1)$ is minimum.*