Linear Layouts Revisited: Stacks, Queues, and Exact Algorithms

Thomas Depian
Algorithms and Complexity Group, TU Wien, Austria

Simon D. Fink
Algorithms and Complexity Group, TU Wien, Austria

Robert Ganian
Algorithms and Complexity Group, TU Wien, Austria

Vaishali Surianarayanan
University of California at Santa Barbara, CA, USA

Abstract

In spite of the extensive study of stack and queue layouts, many fundamental questions remain open concerning the complexity-theoretic frontiers for computing stack and queue layouts. A stack (resp. queue) layout places vertices along a line and assigns edges to pages so that no two edges on the same page are crossing (resp. nested). We provide three new algorithms which together substantially expand our understanding of these problems:

1. A fixed-parameter algorithm for computing minimum-page stack and queue layouts w.r.t. the vertex integrity of an n-vertex graph G. This result is motivated by an open question in the literature and generalizes the previous algorithms parameterizing by the vertex cover number of G. The proof relies on a newly developed Ramsey pruning technique. Vertex integrity intuitively measures the vertex deletion distance to a subgraph with only small connected components.
2. An $n^{\mathcal{O}(q\ell)}$ algorithm for computing ℓ-page stack and queue layouts of page width at most q. This is the first algorithm avoiding a double-exponential dependency on the parameters. The page width of a layout measures the maximum number of edges one needs to cross on any page to reach the outer face.
3. A $2^{\mathcal{O}(n)}$ algorithm for computing 1-page queue layouts. This improves upon the previously fastest $n^{\mathcal{O}(n)}$ algorithm and can be seen as a counterpart to the recent subexponential algorithm for computing 2-page stack layouts [ICALP'24], but relies on an entirely different technique.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases stack layouts, queue layouts, parameterized algorithms, vertex integrity, Ramsey theory

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.15

Related Version *Full Version*: `https://arxiv.org/abs/2508.16319` [14]

Funding *Thomas Depian*: Project No. 10.47379/ICT22029 of the Vienna Science Foundation (WWTF).
Simon D. Fink: Project No. 10.47379/ICT22029 of the Vienna Science Foundation (WWTF) and Project No. 10.55776/Y1329 of the Austrian Science Fund (FWF).
Robert Ganian: Project No. 10.47379/ICT22029 of the Vienna Science Foundation (WWTF) and Projects No. 10.55776/Y1329, 10.55776/COE12 of the Austrian Science Fund (FWF).

1 Introduction

A linear layout $\langle \prec, \sigma \rangle$ of a graph G is a total ordering $\prec$ of its vertices and a partitioning σ of its edges into k *pages* such that each page satisfies certain conditions. The two by far most commonly studied types of linear layouts are *stack layouts* and *queue layouts*; in the former, we require that no four vertices $a \prec b \prec c \prec d$ have the edges ac and bd placed on the same page, while for the latter we forbid any page from containing the edges ad and bc. Intuitively, this corresponds to forbidding edges crossing and edges nesting (in a rainbow pattern), respectively – see Figure 1.

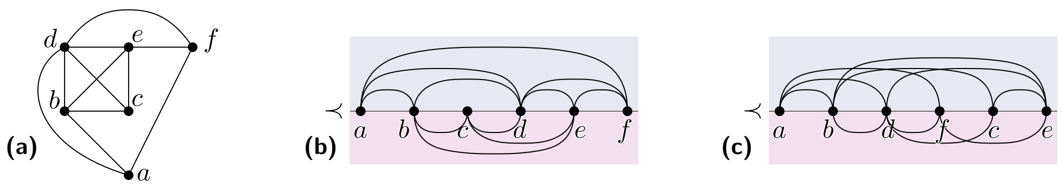


Figure 1 A graph G **(a)** together with a 2-page stack layout **(b)** and 2-page queue layout **(c)**. The two pages are colored blue and lilac, respectively.

Originally motivated from applications in VLSI [10, 41] and bioinformatics [31], stack and queue layouts have become the focus of extensive research. While there are many classical works studying the structural properties of the two notions [17–21, 32, 34, 46], several more recent studies have targeted exact and parameterized algorithms for computing stack and queue layouts that minimize the number of used pages. These results are inherently aimed at circumventing the long-known intractability of these problems: the NP-hardness of determining whether an input graph admits a 2-page stack layout or a 1-page queue layout has been established over thirty years ago [34, 46].

To that end, recent works have pursued the use of structural graph parameters to establish tractability on graphs which are "well-structured". Computing ℓ-page stack layouts is now known to be fixed-parameter tractable w.r.t. the vertex cover number [7] or, alternatively, the feedback edge number of the input graph G [26]. Computing ℓ-page queue layouts is likewise known to admit a fixed-parameter algorithm w.r.t. the vertex cover number of G [8]. One drawback of these results is that they only yield tractability under highly restrictive graph parameters, in the sense of achieving low values only on very "simple" graphs. While computing 1-page queue layouts is also known to be fixed-parameter tractable when parameterized by the treedepth of G [8], for general ℓ it was repeatedly posed as an open question whether the aforementioned vertex-cover based algorithms can be lifted to less restrictive structural graph parameters [7, 8, 21, 25, 26]. Another recent direction is the establishment of tighter algorithmic upper bounds for special cases: while ℓ-page stack and queue layouts can both be computed in time $n^{\mathcal{O}(n)}$ via trivial brute-force algorithms, 2-page stack layouts are now known to admit a subexponential algorithm [26].

Contributions. As our **first contribution**, we lift the aforementioned fixed-parameter tractability of computing linear layouts to a less restrictive structural graph parameter:

▶ **Theorem 1.** *Computing a minimum-page stack layout and minimum-page queue layout is fixed-parameter tractable w.r.t. the* vertex integrity *of the input graph.*

The vertex integrity $\mathsf{vi}(G)$ of a graph G measures, roughly speaking, how many vertex deletions are required to decompose G into small connected components, and has been used in the design of parameterized algorithms for a variety of challenging problems [9, 22–24, 27, 29, 30, 39].[1] More precisely, $\mathsf{vi}(G)$ is the minimum integer k satisfying the following:

Due to space constraints, we defer full proofs of statements marked with ★ to the full version [14].
[1] Some prior works use the *fracture number*, which is parametrically equivalent to vertex integrity.

there exists a vertex set $X \subseteq V(G)$ such that for each connected component C of $G - X$, $|V(C) \cup X| \leq k$. As vertex integrity is upper-bounded by the vertex cover number plus one, Theorem 1 generalizes the fixed-parameter tractable algorithms for computing stack and queue layouts w.r.t. the vertex cover number by Bhore, Ganian, Montecchiani, and Nöllenburg [7, 8]. Moreover, since the vertex integrity is sandwiched between the vertex cover number and treedepth, our result can be seen as a stepping stone towards solving the problem on more general parameterizations.

To establish Theorem 1, we introduce a novel proof technique we call *Ramsey pruning*. On a high level, the algorithm underlying the computation is very simple: assuming $|G|$ is larger than some pre-designated function k, it recursively identifies a connected component C of $G - X$ such that $G - X$ contains sufficiently many copies of C, and deletes C from the instance. The difficulty lies in proving that this operation is safe, i.e., that $G - C$ does not admit a layout with fewer pages than G. The “usual” approach for such a proof would be to show that C can be reinserted into any hypothetical linear layout $\langle \prec, \sigma \rangle$ of $G - C$ without increasing the number of pages; however, this not only seems excruciatingly difficult to prove, but we also believe it to be false for certain choices for $\langle \prec, \sigma \rangle$. Ramsey pruning avoids this issue by using Ramsey-type arguments to argue that $\langle \prec, \sigma \rangle$ must contain a *guiding sub-layout* $\langle \prec', \sigma' \rangle$ – a linear layout of some carefully selected subgraph of $G - C$ – with certain well-defined properties. We then use the well-formedness of $\langle \prec', \sigma' \rangle$ to build a brand new linear layout $\langle \prec^*, \sigma^* \rangle$ of G, thus establishing that G and $G - C$ indeed require the same number of pages. We believe this new technique to be generic and applicable to other problems under the same parameterization, as suggested by the fact that unlike the previous vertex-cover based algorithms our approach works for both stack and queue layouts with almost no problem-specific changes required.

While Theorem 1 pushes the frontiers of tractability for our two problems of interest in a complexity-theoretic sense, our use of Ramsey-type arguments means that the obtained running time bound has an entirely impractical dependency on the parameter (see Lemma 5). In fact, up to now none of the known parameterized algorithms for computing ℓ-page stack or queue layouts have a runtime parameter dependency that would be better than double-exponential – the parameterized algorithms w.r.t. the vertex cover [7, 8] and feedback edge numbers [26] all preprocess the graph to obtain an equivalent instance (a *kernel*) whose size is exponential in the parameter, and then solve that equivalent bounded-size instance via brute force. As our **second contribution**, we provide the first single-exponential algorithms (in the parameters) capable of solving these problems for arbitrary fixed choices of ℓ:

▶ **Theorem 2 (★).** *Given integers ℓ and q along with an n-vertex graph G, we can compute an ℓ-page stack or queue layout with page width at most q of G (if one exists) in time $n^{\mathcal{O}(q \cdot \ell)}$.*

Here, the page width measures the maximum number of edges one needs to cross to reach any vertex from the outer face (see also Section 2). The page width of linear layouts[2] has been studied in several early works [10, 32, 43, 44] and it is generally desirable to obtain layouts not only using few pages, but also with low page width – in fact, the original papers introducing stack layouts explicitly targeted algorithms optimizing both measures. However, the algorithmic applications of page width as a parameter have only been investigated in a recent paper on extending incomplete linear layouts [12]. We remark that while Theorem 2 merely provides a so-called XP *algorithm* in a complexity-theoretic sense, it may still be more efficient in practice than known fixed-parameter algorithms for the problem – especially if the aim is to compute layouts with low pagewidth.

[2] The term *cutwidth* has been used interchangeably with page width in the past [10]; here, we use the latter in order to disambiguate from the related graph parameter cutwidth.

A natural approach towards establishing Theorem 2 would be to use dynamic programming in order to construct the sought-after linear layout in a “left-to-right” fashion; however, the issue is that doing this directly would require us to store roughly 2^n possible subsets of previously processed vertices. To circumvent this, we obtain new insights into the decomposability of layouts with bounded page width, allowing us to construct an $n^{\mathcal{O}(q\cdot\ell)}$-size auxiliary *state graph* H, where we show that the dynamic computation of a sought-after layout can be represented as an easily computable path in H.

As our final **third contribution**, we take a step back from the multivariate analysis of these problems and recall that the trivial $n^{\mathcal{O}(n)}$ barrier was only recently overcome for computing 2-page stack layouts [26], i.e., the lowest-page stack layouts giving rise to an NP-complete problem. Here, we show that the trivial $n^{\mathcal{O}(n)}$ barrier can also be overcome when aiming for the lowest-page queue layouts which still give rise to NP-completeness:

▶ **Theorem 3 (★).** *Given an n-vertex graph G, we can compute a 1-page queue layout of G (if one exists) in time $2^{\mathcal{O}(n)}$.*

We remark that the running time bound provided by Theorem 3 is worse than the $2^{\mathcal{O}(\sqrt{n})}$ bound previously obtained for 2-page stack layouts [26]. The reason for this is that the latter result relied on the equivalence of 2-page stack layouts with the existence of *subhamiltonian paths*; the authors of that previous work then essentially obtained a single-exponential fixed-parameter algorithm for finding such paths w.r.t. the graph parameter treewidth, improving on the previously established fixed-parameter tractability of the problem [4].

However, emulating the same approach seems difficult in the queue layout setting: 1-page queue layouts also have an equivalent formulation in terms of so-called *arched-level planar* drawings [34], but it is not at all clear whether computing such drawings is fixed-parameter tractable w.r.t. treewidth (not to mention the fact that one would need a single-exponential algorithm). Instead, our proof of Theorem 3 relies on a Turing reduction from computing arched-level planar drawings to $2^{\mathcal{O}(n)}$ many instances of a well-studied problem called LEVEL PLANARITY, which is known to be solvable in polynomial time [33, 37].

Related Work. We refer to the dedicated survey for an overview of many earlier structural results concerning linear layouts [19]. Key structural results in the area include the existence of 4-page stack layouts [46] and 42-page queue layouts [5, 17] for all planar graphs. Researchers have also studied the notion of *mixed layouts*, which are linear layouts with some pages behaving like stack and some like queue layouts [3, 38]. We note that due to the techniques used in their proofs, it seems very likely that Theorems 1 and 2 could be adapted to the mixed layout setting with minimum changes required. Finally, we remark that parameterized algorithms for stack and/or queue layouts have additionally been considered in the extension setting (where the task is to complete a provided partial layout) [12, 13], in the upward-planarity setting (where the graph is directed and edges must be oriented in a left-to-right fashion along the layout) [6] and when the vertex ordering in the layout is fixed [1, 40].

2 Preliminaries

For an integer $\ell \geq 1$ we let $[\ell]$ denote the set $\{1, 2, \dots, \ell\}$. We assume the reader to be familiar with standard graph terminology [15]. Without loss of generality, we assume all input graphs to be connected, have n vertices and m edges. For a set of vertices $V' \in V(G)$, we let $G[V']$ denote the graph induced on V'. Furthermore, for a set of edges $E' \in E(G)$, we let $V(E')$ denote the set of its endpoints and $G[E'] = (V(E'), E')$. A *cut* (A, B) of G is a partition of the vertices in $A \subseteq V(G)$ and $B = V(G) \setminus A$. We call $F = \{uv \in E(G) \mid u \in A, v \in B\}$ a *cut-set* of *size* $|F|$ that *induces* the cut (A, B). Throughout the paper, we omit an explicit reference to G if it is clear from the context, e.g., we write V and E instead of $V(G)$ and $E(G)$.

Given a linear order $\prec$ of a graph G and two vertices $u, v \in V$, we say that u is *left* of v if $u \prec v$ and *right* of v if $v \prec u$. The vertices u and v are *consecutive on the spine* if they occur consecutively in $\prec$, i.e., there is no vertex $w \in V$ such that $u \prec w \prec v$ or $v \prec w \prec u$ holds. We address with $\mathrm{Left}_v^{\prec} \coloneqq \{u \in V \mid u \prec v\}$ all vertices $u \in V$ left of v with respect to $\prec$ and define $\mathrm{Right}_v^{\prec} \coloneqq V \setminus (\mathrm{Left}_v^{\prec} \cup \{v\})$. Finally, we let $X_v^{\prec} \coloneqq \{uw \in E \mid u \in \mathrm{Left}_v \cup \{v\}, w \in \mathrm{Right}_v\}$ denote the set of edges that *span* the spine between v and its right neighbor, i.e., all edges with one endpoint left of (or at) v and one right of v.

We assume familiarity with the basic foundations of parameterized complexity theory [11]. All algorithms obtained in this work are exact, deterministic, constructive and rely exclusively on computable functions. To express some of our bounds, we will occasionally use the Knuth notation $\uparrow\uparrow$ where for an integer z, $2 \uparrow\uparrow z$ represents an exponential tower of 2's of height z.

Linear Layouts. Let u_1v_1 and u_2v_2 be two edges of G. We say that they *cross* under a linear order $\prec$ if $u_1 \prec u_2 \prec v_1 \prec v_2$ holds, and they *nest* if $u_1 \prec u_2 \prec v_2 \prec v_1$ holds. For an integer $\ell \geq 1$, let $\sigma_G \colon E(G) \to [\ell]$ be a function that assigns each edge to a *page* $p \in [\ell]$. The *linear layout* $\langle \prec_G, \sigma_G \rangle$ is a *stack* (*queue*) *layout* if no two edges $e_1, e_2 \in E$ with $\sigma_G(e_1) = \sigma_G(e_2)$ cross (nest). We call $\prec_G$ the *spine order* and σ_G the *page assignment*. For the remainder of the paper, we write $\prec$ and σ if the graph G is clear from context.

The *page width* $\omega(\langle \prec, \sigma \rangle)$ of an ℓ-page linear layout $\langle \prec, \sigma \rangle$ corresponds to the maximum number of edges on a single page that span the spine between a vertex and its right neighbor, i.e., $\omega(\langle \prec, \sigma \rangle) \coloneqq \max_{p \in [\ell]} \max_{v \in V(G)} \left| X_v \cap \sigma^{-1}(p) \right|$. We call an ℓ-page linear layout with page width q an *ℓ-page q-width linear layout*, or simply a *solution* when the ℓ and q are clear from context. In line with the literature, we assume no explicit bound on the page width when q is not specified.

Vertex Integrity. A graph G has *vertex integrity* $\mathsf{vi}(G) = p$ if p is the smallest integer with the following property: G contains a vertex set S such that for each connected component H of $G - S$, $|V(H) \cup S| \leq p$. One may observe that the vertex integrity is upper-bounded by the size of a minimum vertex cover in the graph (i.e., the vertex cover number) plus one. The vertex integrity of an n-vertex graph along with a corresponding partition into S and $\mathcal{C} = G - S$ can be computed in time $\mathcal{O}(p^{p+1} \cdot n)$ [16].

3 A Fixed-Parameter Algorithm Parameterized by Vertex Integrity

In this section, we obtain a fixed-parameter algorithm that takes as input a positive integer ℓ and a graph G, is parameterized by $\ell + \mathsf{vi}(G)$ and computes an ℓ-page stack and/or queue layout of G (or both), if such layouts exist. We begin by noting that using well-known relationships between vertex integrity, minimum-page stack and queue layouts and the graph parameter *treewidth* (tw), we can assume ℓ to be upper-bounded by a function of $\mathsf{vi}(G)$ thanks to the well-known relation $\mathsf{tw}(G) \leq \mathsf{vi}(G)$:

▶ **Proposition 4** ([20, 45]). *The number of pages in a minimum stack and queue layout of a graph G is upper-bounded by $\mathsf{tw}(G) + 1 \leq \mathsf{vi}(G) + 1$ and $2^{\mathsf{tw}(G)} + 1 \leq 2^{\mathsf{vi}(G)} + 1$, respectively.*

The algorithm operates by obtaining a problem kernel, as formalized below; recall $p = \mathsf{vi}(G)$.

▶ **Lemma 5.** *There is an $\mathcal{O}(p^{p+1} \cdot n)$ time algorithm that takes a graph G with an integer ℓ and outputs a subgraph G' of G (a* kernel*) of size at most $\left(2 \uparrow\uparrow (p \cdot 2^{2p^2}) \cdot 2 + 6\right)^{\ell \cdot p}$ with the following property: G' admits an ℓ-page stack (or queue) layout $\langle \prec_{G'}, \sigma_{G'} \rangle$ if and only if so does G. Moreover, such a layout for G can be computed from $\langle \prec_{G'}, \sigma_{G'} \rangle$ in polynomial time.*

Lemma 5 directly implies Theorem 1, and the rest of this section is dedicated to proving this lemma. For the following, let us fix a graph G, positive integers ℓ and p, and a choice of $S \subseteq V(G)$ witnessing $\mathsf{vi}(G) = p$ as in Section 2. Further let $\mathcal{C}$ be the set of connected components of $G - S$. First, we define a notion of "component-types" which groups components in $\mathcal{C}$ that exhibit the same outside connections and internal structure.

▶ **Definition 6.** *We say two graphs $H_0, H_1 \in \mathcal{C}$ are* twins*, denoted $H_0 \sim H_1$, if there exists a canonical isomorphism α from H_0 to H_1 such that for each vertex $u \in V(H_0)$ and each $v \in S$, $uv \in E(G)$ if and only if $\alpha(u)v \in E(G)$.*

▶ **Lemma 7 (★).** *Each graph $H \in \mathcal{C}$ has at most p vertices, $\sim$ is an equivalence relation and the number of equivalence classes in $[\sim]$ is upper-bounded by $p \cdot 2^{2p^2}$. Moreover, a partition of connected components into $[\sim]$ can be computed in time at most $\mathcal{O}(p \cdot 2^{2p^2} \cdot n)$.*

We now introduce the notion of large equivalence classes based on their size. We then use this to define what we call a large group of vertices – one that contains exactly one representative from each large equivalence class. Our kernel will keep a bounded number of these large groups.

▶ **Definition 8.** *Let k be a positive integer, an equivalence class $[H]$ of $\sim$ is said to be k-*large *if $|[H]| \geq k$. Further a vertex set $L \subseteq V(G)$ is called a k-*large group *if the induced subgraph $G[L]$ is a disjoint union of exactly one graph from each k-large equivalence class of $\sim$.*

Next we define a special induced subgraph that will serve as our kernel. The definition is based on carefully choosing a k that is bounded by a computable function of p and ℓ so that keeping only k many k-large groups along with the small parts of the graph suffices to capture the necessary structure. Towards this, let us first fix $f(\ell, p, x) := 2^{2^{\ell \cdot x^2 \cdot 2^{12p^2}}}$ to be a computable function that is large enough to apply our Ramsey-type arguments later on; here, x will be an integer that represents the size of a deletion set (initially S, but this will be updated iteratively in the proof of Lemma 11). Moreover, let $g(\ell, p)$ be a computable function of ℓ and p that will upper-bound our nested application of the function f in that same proof; to provide a concrete bound, we set $g(\ell, p) := \left(2 \uparrow\uparrow (p \cdot 2^{2p^2}) \cdot 2 + 4\right)^{\ell \cdot p}$.

▶ **Definition 9.** *An induced subgraph G' of G is said to be a* reduced graph *of G if there exists a positive integer $x \leq g(\ell, p)$ and a partition of $V(G') = S \uplus S' \uplus Y$ satisfying:*

1. *$|S \cup S'| = x$ and S' is the set of all vertices in graphs in $\mathcal{C}$ that do not belong to an $f(\ell, p, x)$-large equivalence class of $\sim$.*
2. *If there are no $f(\ell, p, x)$-large equivalence classes of $\sim$, $Y = \emptyset$ and $V(G') = S \uplus S' = V(G)$. Otherwise, it holds that $Y = L_1 \uplus \cdots \uplus L_{f(\ell,p,x)}$ where L_i is an $f(\ell, p, x)$-large group for each $i \in \{1, \cdots, f(\ell, p, x)\}$, and each pair of L_i and L_j, $i \neq j$, is vertex-disjoint.*

Intuitively, the reduced graphs we will be dealing with will consist of S, a set S' of all equivalence classes of $\sim$ which are too small to fully saturate our large groups (these will later be treated essentially in the same way as S), and a sufficient number of large groups; equivalence classes of size larger than $f(\ell, p, x)$ are "pruned" to have size exactly $f(\ell, p, x)$. A schematic overview of this intuition can be found in Figures 2b and c later on.

The core of our result is the following lemma, which we prove separately in Section 3.1.

▶ **Lemma 10.** *If G' is a reduced graph of G then G has an ℓ-page stack (queue) layout if and only if G' has an ℓ-page stack (queue) layout.*

Before proceeding to that proof, we show how to construct a reduced graph having size bounded by a computable function of p and ℓ in polynomial time.

▶ **Lemma 11 (★).** *There exists a reduced graph G' of G, and given S and $\sim$ such a graph can be computed in polynomial time. Further, the number of vertices in G' is upper-bounded by $\left(2 \uparrow\uparrow (p \cdot 2^{2p^2}) \cdot 2 + 6\right)^{\ell \cdot p}$.*

Lemma 11 combined with Lemma 10 establishes Lemma 5 by constructing a reduced graph which is a kernel of the desired size. Thus, what remains is to establish Lemma 10; this is where the core ideas of Ramsey pruning will come into play.

3.1 Proof of Lemma 10

For this proof, we fix G' to be a reduced graph of G. Since G' is an induced subgraph of G, it is clear that if G has an ℓ-page stack (queue) layout then G' also has an ℓ-page stack (queue) layout. To complete the proof we show that the reverse is true: If G' has an ℓ-page stack (queue) layout, then G also has an ℓ-page stack (queue) layout.

Recall that $f(\ell, p, x) := 2^{2^{\ell \cdot x^2 \cdot 2^{12p^2}}}$. Let $\langle \prec, \sigma \rangle$ be a fixed (but hypothetical) ℓ-page stack (queue) layout of G'; throughout this section, for $H \subseteq G'$ we will use $\langle \prec_H, \sigma_H \rangle$ to refer to the sublayout of $\langle \prec, \sigma \rangle$ induced on the subgraph H of G'. If $V(G') = V(G)$ we are done. So let $V(G') = S \uplus S' \uplus L_1 \uplus \cdots \uplus L_{f(\ell,p,|S\cup S'|)}$ be a partition witnessing that G' is a reduced graph. Here each L_i is a $f(\ell, p, |S \cup S'|)$-large group and S' is the set of all vertices in graphs that do not belong to an $f(\ell, p, |S \cup S'|)$-large equivalence class in $\sim$. For brevity, we will hereinafter use *large* as shorthand for $f(\ell, p, |S \cup S'|)$-large.

Let $\mathcal{L} = \{L_1, \cdots, L_{f(\ell,p,|S\cup S'|)}\}$. Our key idea is to identify three large groups $X, Y, Z \in \mathcal{L}$ with a special structure (a "guiding sublayout") in the solution $\langle \prec, \sigma \rangle$ using Ramsey theory. We will then use this pattern to insert the remaining parts of the graph. Before proceeding, we define some notations to be able to identify these groups, starting with a "template" R.

▶ **Definition 12.** *Let k be the number of large equivalence classes in $\sim$ and let $R_i \in \mathcal{C}$ be a canonical representative of the i^{th} large equivalence class $[R_i]$. Let $R := V(R_1) \uplus \cdots \uplus V(R_k)$.*

Recall that by Definition 8, if X is a large group then $G[X]$ is a disjoint union of exactly one graph from each large equivalence class of $\sim$. We now define a natural isomorphism from $G[X]$ to $G[R]$. This will allow us to map consistently between different large groups via R.

▶ **Definition 13.** *For a large group X with $G[X] = H_1 \uplus \cdots \uplus H_k$, $H_i \in [R_i]$ for each $i \in [k]$, let α_X be an isomorphism from $G[X]$ to $G[R] = R_1 \uplus \cdots \uplus R_k$ such that for each $i \in [k]$ and vertex $u \in H_i$, $\alpha_X(u) \in V(R_i)$, and for each $v \in S$, $uv \in E(G)$ if and only if $\alpha_X(u)v \in E(G)$.*

For any large group A and for each $u \in R$ we will sometimes use u_A as shorthand for $\alpha_A^{-1}(u)$. From now we fix two distinct large groups L and L' in $\mathcal{L}$. Further, for distinct large groups $X, Y \in \mathcal{L}$, we say $X \prec Y$ if in $\prec$, the first vertex in $X \cup Y$ is from X.

▶ **Definition 14.** *For $X \prec Y \in \mathcal{L} \setminus \{L, L'\}$, we define $\phi_{X,Y} : S \cup S' \cup X \cup Y \to S \cup S' \cup L \cup L'$*

- *$\phi_{X,Y}(s) = s$ for each $s \in S \cup S'$*
- *$\phi_{X,Y}(x) = \alpha_L^{-1}(\alpha_X(x))$ for each $x \in X$*
- *$\phi_{X,Y}(y) = \alpha_{L'}^{-1}(\alpha_Y(y))$ for each $y \in Y$*

Note that $\phi_{X,Y}$ is an isomorphism from $G[S \cup S' \cup X \cup Y]$ to $G[S \cup S' \cup L \cup L']$.

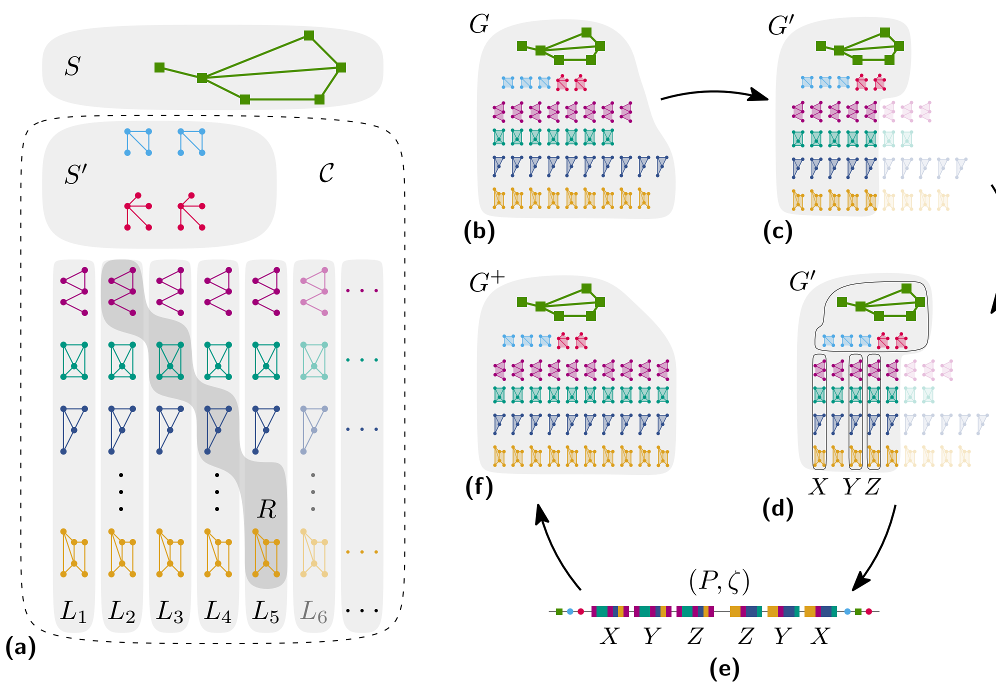


Figure 2 A schematized overview of **(a)** the used notation and **(b)**–**(f)** proof of Theorem 1. In the latter, G is the original input graph, G' the reduced graph obtained after applying Lemma 11, and $S \cup S' \cup X \cup Y \cup Z$ form the guiding sublayout that is used to construct an ℓ-page layout for a graph G^+ which is a supergraph of G – hence establishing that G' admits an ℓ-page stack (or queue) layout if and only if so does G.

▶ **Definition 15.** *For $X \prec Y \in \mathcal{L} \setminus \{L, L'\}$, $\mathsf{info}_{\langle \prec,\sigma\rangle}(X, Y)$ is the stack (queue) layout of $G[S \cup S' \cup L \cup L']$ obtained from $\langle \prec_{G[S\cup S'\cup X\cup Y]}, \sigma_{G[S\cup S'\cup X\cup Y]}\rangle$ using the isomorphism $\phi_{X,Y}$.*

We are now ready to show the existence of three distinct large groups X, Y, Z with a useful consistent pattern between them and $S \cup S'$ based on info. Essentially, these three groups will form the aforementioned guiding sublayout used to argue the correctness of the pruning step, i.e., establish Lemma 10 by showing that we can reinsert all the removed vertices by building on $\langle \prec_{G[S\cup S'\cup X\cup Y\cup Z]}, \sigma_{G[S\cup S'\cup X\cup Y\cup Z]}\rangle$. Note that – perhaps counterintuitively – we will do so by discarding all other information about $\langle \prec, \sigma\rangle$. We refer to Figure 2 for an overview of the entire approach and to Figure 4 for a visualization of info.

▶ **Lemma 16.** *There exists distinct large groups $X, Y, Z \in \mathcal{L} \setminus \{L, L'\}$ with $X \prec Y \prec Z$ such that $\mathsf{info}_{\langle \prec,\sigma\rangle}(X, Y) = \mathsf{info}_{\langle \prec,\sigma\rangle}(Y, Z) = \mathsf{info}_{\langle \prec,\sigma\rangle}(X, Z)$.*

Proof. Construct a complete edge colored auxiliary graph $\mathcal{H}$ with $V(\mathcal{H}) = \mathcal{L} \setminus \{L, L'\}$. For $X, Y \in \mathcal{L} \setminus \{L, L'\}$ with $X \prec Y$, we set $\mathit{color}((X, Y)) = \mathsf{info}_{\langle \prec,\sigma\rangle}(X, Y)$. Let $x := |S \cup S'|$.

▷ Claim 17 (★). The number of possible distinct edge colors of $\mathcal{H}$ is at most $2^{\ell\cdot x^2\cdot 2^{9p^2}}$.

We now use a well-known fact (from Ramsey theory [2]) that any edge-colored clique on n vertices colored with t colors has a monochromatic clique of size at least $\log_t(n)/t$.

We have $|V(\mathcal{H})| = |\mathcal{L}| - 2 = f(\ell, p, x) - 2 \geq 2^{2^{\ell\cdot x^2\cdot 2^{11p^2}}}$. Thus $n \geq 2^{2^{\ell\cdot x^2\cdot 2^{11p^2}}}$ and $t \leq 2^{\ell\cdot x^2\cdot 2^{9p^2}}$ and therefore there is a monochromatic clique of size at least $\log_t(n)/t \geq 2^{2^2} \geq 3$. A monochromatic clique of size 3 in $\mathcal{H}$ corresponds to distinct large groups $X, Y, Z \in \mathcal{L}\setminus\{L, L'\}$ with $X \prec Y \prec Z$ such that $\mathsf{info}_{\langle \prec,\sigma\rangle}(X, Y) = \mathsf{info}_{\langle \prec,\sigma\rangle}(Y, Z) = \mathsf{info}_{\langle \prec,\sigma\rangle}(X, Z)$. ◀

Let $X, Y, Z \in \mathcal{L}$ be three large groups witnessing the previous lemma. We now show that we can partition the vertices of X, Y, and Z in a nice way that will guide us on how to insert the remaining parts of the graph. Towards this, we first note that the $\mathsf{info}_{\langle \prec, \sigma \rangle}$ forces (1) edges incident to X, Y and Z to have the same page assignment, and moreover (2) forces the individual vertex orderings of $X \cup S \cup S'$, $Y \cup S \cup S'$, $Z \cup S \cup S'$ to be exactly the same as some ordering $\prec'$ of $R \cup S \cup S'$; see also the full version [14, Observations 18 and 19]. However, this does not yet fix how the vertices of X, Y and Z are ordered with respect to each other.

For the following, let us set $\Upsilon = S \cup S' \cup X \cup Y \cup Z$. Let a *block* be a consecutive subsequence of vertices from R w.r.t. $\prec'$, and a *solution-block* of $\langle \prec_{G[\Upsilon]}, \sigma_{G[\Upsilon]} \rangle$ be a consecutive subsequence of vertices from Υ; see also Figure 3.

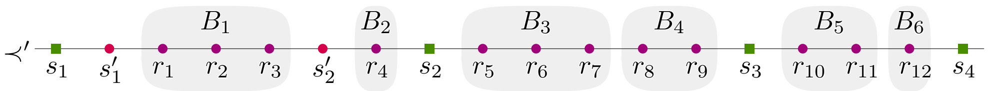


Figure 3 An ordering $\prec'$ of $R \cup S \cup S'$ and a partition of R into the blocks B_1 to B_6, indicated in gray. Note that vertices of $S \cup S'$, colored green and red, respectively, always separate two blocks. However, not every pair of blocks is separated by such a vertex, see, for example, blocks B_3 and B_4.

We are now ready to prove the structural result which provides strong guardrails on $\langle \prec_{G[\Upsilon]}, \sigma_{G[\Upsilon]} \rangle$; in particular, it guarantees that all vertices from X, Y and Z must occur in consecutive solution-blocks and following a strict "ascending" or "descending" order. We remark that this structural result is only made possible by the extraction of three copies of large groups via Lemma 16; see also Figure 4 for an illustration.

▶ **Lemma 18 (★).** *There exists a pair (P, ζ), where P is a partitioning of R into blocks and $\zeta \colon P \to \{\nearrow, \searrow\}$ satisfying the following. For every block $B \in P$ with $\zeta(B) = \nearrow$, the sequence $\alpha_X^{-1}(B)\alpha_Y^{-1}(B)\alpha_Z^{-1}(B)$ is a solution-block of $\langle \prec_{G[\Upsilon]}, \sigma_{G[\Upsilon]} \rangle$. Moreover, for every block $B \in P$ with $\zeta(B) = \searrow$, the sequence $\alpha_Z^{-1}(B)\alpha_Y^{-1}(B)\alpha_X^{-1}(B)$ is a solution-block of $\langle \prec_{G[\Upsilon]}, \sigma_{G[\Upsilon]} \rangle$.*

Crucially, we can now use Lemma 18 to safely insert an arbitrary number of large groups into $G[\Upsilon]$ without increasing the number of required pages.

▶ **Lemma 19 (★).** *Let G^+ be the supergraph of G with each large equivalence class having equal size, then G^+ has an ℓ-page stack (queue) layout.*

Proof sketch. Let $t \geq 3$ be the maximum size of a large equivalence class of $\sim$. Note that G^+ can be expressed as $G^+ = G^+[S \cup S' \cup X_1 \cup \dots \cup X_t]$ where each $X_i, i \in [t]$ is a large group. We construct a stack (or queue, as desired) layout $\langle \prec^+, \sigma^+ \rangle$ of G^+ as follows.

Construction of σ^+. For each edge $(u, v) \in G^+$,

- If $u, v \in S \cup S'$, we set $\sigma^+(u, v) := \sigma(u, v)$.
- If $u \in X_i$ for some $i \in [t]$ and $v \in S$, we set $\sigma^+(u, v) := \sigma(r_Y, v)$ where $r = \alpha_{X_i}(u)$.
- If $u, v \in X_i$ for some $i \in [t]$, we set $\sigma^+(u, v) := \sigma(r_Y, r'_Y)$ where $r = \alpha_{X_i}(u)$, $r' = \alpha_{X_i}(v)$.

Construction of $\prec^+$. In $\prec'$, replace each block $B \in P$ with $\alpha_{X_1}^{-1}(B)\alpha_{X_2}^{-1}(B)\cdots\alpha_{X_t}^{-1}(B)$ if $\zeta(B) = \nearrow$ and with $\alpha_{X_t}^{-1}(B)\alpha_{X_{t-1}}^{-1}(B)\cdots\alpha_{X_1}^{-1}(B)$ if $\zeta(B) = \searrow$.

To show that $\langle \prec^+, \sigma^+ \rangle$ is indeed an ℓ-page stack (queue) layout of G^+, we show that any conflict in $\langle \prec^+, \sigma^+ \rangle$ would also imply a conflict in $\langle \prec_{G[\Upsilon]}, \sigma_{G[\Upsilon]} \rangle$ in the full proof [14]. ◀

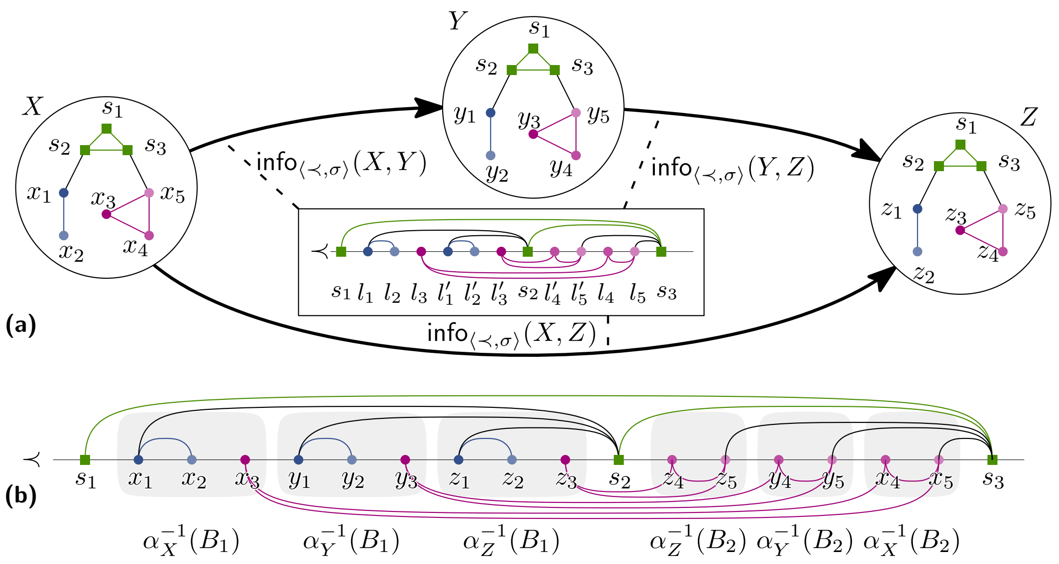

(a)
X
Y
Z
info$_{\langle\prec,\sigma\rangle}(X,Y)$
info$_{\langle\prec,\sigma\rangle}(Y,Z)$
info$_{\langle\prec,\sigma\rangle}(X,Z)$
$s_1\ l_1\ l_2\ l_3\ l'_1\ l'_2\ l'_3\ s_2\ l'_4\ l'_5\ l_4\ l_5\ s_3$
(b)
$s_1\ x_1\ x_2\ x_3\ y_1\ y_2\ y_3\ z_1\ z_2\ z_3\ s_2\ z_4\ z_5\ y_4\ y_5\ x_4\ x_5\ s_3$
$\alpha_X^{-1}(B_1)$ $\alpha_Y^{-1}(B_1)$ $\alpha_Z^{-1}(B_1)$ $\alpha_Z^{-1}(B_2)$ $\alpha_Y^{-1}(B_2)$ $\alpha_X^{-1}(B_2)$

Figure 4 (a) The three large groups $X \prec Y \prec Z \in \mathcal{L} \setminus \{L, L'\}$ form a monochromatic clique in the auxiliary graph $\mathcal{H}$, i.e., we have $\mathsf{info}(X,Y) = \mathsf{info}(Y,Z) = \mathsf{info}(X,Z)$. The tuple (P,ζ), where P partitions R into two blocks $B_1 = \{r_1, r_2, r_3\}$ and $B_2 = \{r_4, r_5\}$ with $\zeta(B_1) = \nearrow$ and $\zeta(B_2) = \searrow$ fulfills the properties from Lemma 18. As X, Y, Z form a monochromatic clique, the solution blocks from **(b)** must occur in $\langle \prec_{G[\Upsilon]}, \sigma_{G[\Upsilon]} \rangle$.

4 An XP-Algorithm For Bounded-Width Linear Layouts

In this section, we present an algorithm to decide if a given graph G has a stack (or queue) layout on ℓ pages and with page width q, i.e., an ℓ-page q-width linear layout, in $n^{\mathcal{O}(q\cdot\ell)}$ time. In the affirmative case, it can also output such a layout. Since only little adaption is required between stack and queue layouts, we only discuss problem specific changes where necessary. We will first give an intuitive overview over our approach and the core ideas to showing its correctness, details can be found in Sections 4.1 and 4.2 and the full version.

Bounded-size Interfaces. Observe that, for each vertex $v \in V$ in a stack (or queue) layout $\langle \prec, \sigma \rangle$ of G, the edges $X_v^{\prec}$ that span the spine right next to v, induce the cut $(\mathrm{Left}_v^{\prec} \cup \{v\}, \mathrm{Right}_v^{\prec})$ in G; see Figure 5a. The size of the cut-set $F = X_v^{\prec}$ only depends on the number of the pages and their width. Our algorithm is centered around the insight that, through limiting these parameters and thus the size of any such cut, we limit the size of the "interface" (a notion we will define more precisely in a second) between a drawing on its left side and a drawing on its right side (Lemma 22). We note that this insight parallels to the approach used by Saxe [42] to test if a graph has bandwidth at most k. In our setting, replacing the left side of such cut in any solution drawing with one that provides the same interface will yield another solution. This is because having the same interface will imply inducing the same decompositions into left and right vertices (Lemma 24), the statement then follows by simply gluing the two drawings together along the cut.

Our bounded-size interface consists of the set of edges F that intersect the cut right next to v, their vertical order along the cut, and the information which edge endpoint lies on which side of the cut, together with a page assignment for the edges in F. We will encapsulate this information in what we call a (nicely) *oriented cut-set*; see Definition 21.

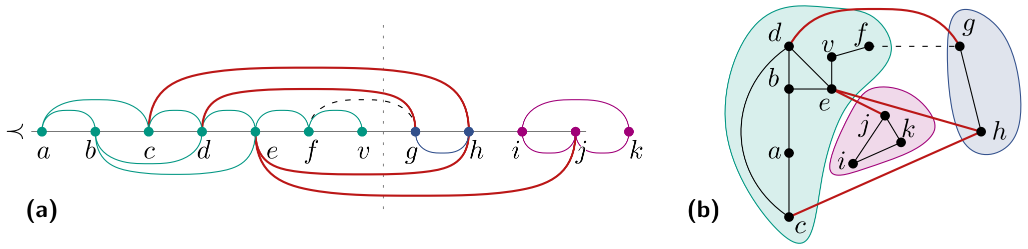


Figure 5 The two-page width-five stack layout in **(a)** induces the cut-set $F = X_v^{\prec}$ indicated with the red edges. **(b)** The three connected components in $G \setminus F$ are indicated with color. Observe that $(F, c \prec d \prec e \prec g \prec h \prec j)$ is nicely oriented. The source e and sink g cannot be in the same connected component, as every path between them contains at least one edge of F; e.g., the edge fg cannot exist.

Our Dynamic Programming Solution. Given only an arbitrary oriented cut-set F for an instance G, we are able to infer which connected components and thus which vertices of the graph lie on the left (or also right) side of the cut (Lemma 23; see also Figure 5b). This now allows us to solve this problem via dynamic programming. We build a directed acyclic state graph where each state corresponds to one configuration of our interface, i.e. one (nicely) oriented cut-set together with a page assignment for its edges (formalized as $\mathcal{N}$C 1 and 2 in the full version [14]). We add an arc between two states if we can move from one oriented cut-set to the other by moving exactly one vertex (which we inferred to be on the right side via Lemma 23) to the (end of a linear order used on the) left without directly violating a necessary stack or queue layout property (i.e., no crossing or nesting of edges) with regard to only the edges in the current cut-set; the exact conditions are again formalized as $\mathcal{A}$C 1–4 in the full version [14]. In Lemma 25, we will show that this means that we can extend any partial solution represented by the current state with the further vertex when moving along such an arc to obtain a solution for the new state. Then, we have a yes-instance if and only if there exists a directed path between the (artificially-distinguished) empty beginning and end states. As finding such path takes linear time, Theorem 2 then follows from the insight that the state graph has a size bounded by $n^{\mathcal{O}(q \cdot \ell)}$ and can be computed in this time.

4.1 Decomposing Linear Layouts Using Oriented Cut-Sets

As already hinted above, our algorithm uses at its core cut-sets of G to compute the desired stack (or queue) layout $\langle \prec_G, \sigma_G \rangle$. Before we discuss their usage in detail, let us first take a closer look at them and define their desired properties.

▶ **Observation 20.** *Given a connected graph G and a cut-set $F \subseteq E$ that separates the graph into $k \geq 2$ connected components $C_1, \ldots, C_k$. For all $i \in [k]$, component C_i contains at least one vertex that is an endpoint of an edge in F.*

We consider in addition to a cut-set F a total order $\prec_F$ on the endpoints of its edges and call this the *oriented cut-set*, denoted as $(F, \prec_F)$. While a single cut-set F has $\mathcal{O}((2\,|F|)!)$ different oriented cut-sets, we are only interested in "nicely oriented" cut-sets. Let F be a cut-set and $\prec_F$ a linear order on the endpoints of edges in F. A vertex $v \in V(F)$ is a *source* (or *sink*) in $(F, \prec_F)$ if for every edge $uv \in F$ we have $v \prec_F u$ ($u \prec_F v$, respectively).

▶ **Definition 21.** *For a cut-set F, $(F, \prec_F)$ is* nicely oriented *if and only if each vertex $v \in V(F)$ is either a source or sink, each connected component of $G - F$ contains either only sources or only sinks, and all sources are left of every sink in $\prec_F$.*

To ease presentation, we use a slight abuse of notation and allow the linear order $\prec_F$ of an oriented cut-set $(F, \prec_F)$ to be a total order of a super-set of $V(F)$. We consider $(F, \prec)$ and $(F, \prec')$ identical if $\prec$ and $\prec'$ agree on $V(F)$. We now establish a crucial connection between linear layouts and nicely oriented cut-sets of G; see also Figure 5.

▶ **Lemma 22 (★).** *Let G be a graph and $\langle \prec, \sigma \rangle$ be an ℓ-page q-width linear layout of G. For every vertex $v \in V$ that is not the rightmost in $\prec$, the edges $X_v^{\prec}$ form a cut-set of size at most $q \cdot \ell$ that induces the cut $(Left_v^{\prec} \cup \{v\}, Right_v^{\prec})$ in G. Furthermore, $(X_v^{\prec}, \prec)$ is nicely oriented.*

A liner order $\prec$ *induces* an oriented cut-set $(F, \prec')$ if there exists a vertex v that witnesses $(X_v^{\prec}, \prec) = (F, \prec')$. Note that the same oriented cut-set can be induced by different linear orders. From Lemma 22, we can deduce that all cuts (A, B) of an ℓ-page q-width stack (and queue) layout of G can be constructed using at most $\binom{m}{\ell \cdot \omega} = \mathcal{O}(m^{\ell \cdot \omega})$ different cut-sets. We aim to use their oriented counterparts to obtain the spine order $\prec$ of the desired stack (or queue) layout. We next show that we can efficiently compute for a given oriented cut-set $(F, \prec_F)$ a witnessing linear order that induces $(F, \prec_F)$.

▶ **Lemma 23 (★).** *Given a graph G and a nicely oriented cut-set $(F, \prec_F)$ of G, we can compute a linear order $\prec$ of V that induces $(F, \prec_F)$ in $\mathcal{O}(n + m)$ time.*

Lemma 23 only shows that we can compute *some* linear order $\prec$ of V that induces $(F, \prec_F)$. However, to efficiently use oriented cut-sets in our state graph, we must be able to precisely determine which vertices have already been placed on the spine without storing them explicitly. The following lemma lays the foundation for this, as it shows that no matter which linear order $\prec$ we choose to induce an oriented cut-set $(F, \prec_F)$, we obtain the same partition into left and right vertices.

▶ **Lemma 24 (★).** *Let $(F, \prec_F)$ be a nicely oriented cut-set. Furthermore, let $\prec$ and $\prec'$ be two linear orders such that $(X_v^{\prec}, \prec) = (X_u^{\prec'}, \prec') = (F, \prec_F)$ for some $u, v \in V$, i.e., they induce $(F, \prec_F)$. Then we have $Left_v^{\prec} \cup \{v\} = Left_u^{\prec'} \cup \{u\}$.*

4.2 Using Nicely Oriented Cut-Sets to Show Theorem 2

The state graph H contains a node for each possible nicely-oriented cut-set of G, together with the corresponding page assignment. The arcs in H represent possible transitions from one nicely oriented cut-set to another that can occur when traversing a hypothetical solution from left to right (see Figure 6), i.e., that are obtained by moving exactly one vertex from the right to the left. A state $S = (F, \prec_{F_X}, \sigma_{F_X})$ is *feasible* if and only if there exists an ℓ-page q-width stack layout $\langle \prec_S, \sigma_S \rangle$ of $G[L(S) \cup V(F)]$ such that for all $u, v \in V(F)$ we have $u \prec_F v \Leftrightarrow u \prec_S v$ and $\sigma_F = \sigma_S|_F$. For queue layouts, the definition is analogous. In the full version, we show that, essentially, having arcs only between state pairs where both induce valid stack (or queue) layouts on their respective cut-sets is enough to preserve feasibilty along arcs.

▶ **Lemma 25 (★).** *Let $H = (N, A)$ be a state graph and $S_X S_Y \in A$ one of its arcs. If S_X is feasible for stack (queue) layouts then so is S_Y.*

We now have all tools at hand to present our algorithm. Since the arcs of H preserve existence of a (partial) ℓ-page q-width stack (or queue) layout of G, we have reduced the problem of finding such a layout to the problem of finding a path in H between the two special nodes $S_\emptyset$ and S_V. In the full version, we show that H can be computed in polynomial time and use this to summarize the main result of this section in the following theorem.

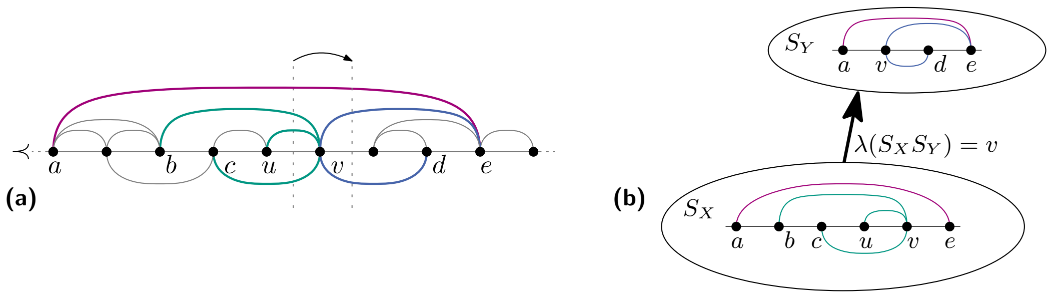


Figure 6 (a) Parts of a stack layout of G with the two cut-sets $X_u^{\prec}$ and $X_v^{\prec}$. **(b)** The corresponding nodes in the state graph. The arc in (b) corresponds to the transition from $X_u^{\prec}$ to $X_v^{\prec}$ in (a). Edges appearing in at least one cut-set are colored.

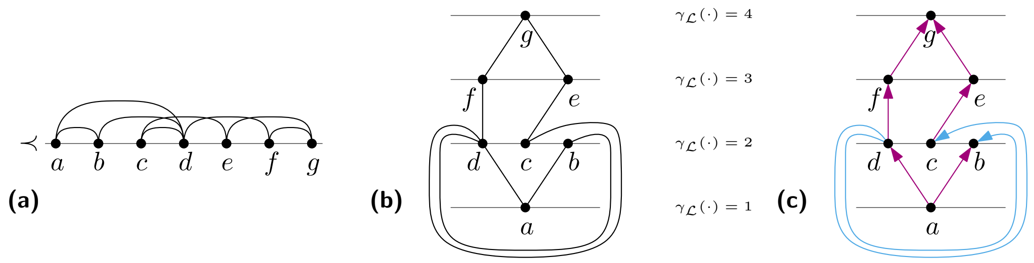
▶ **Theorem 2** (★)**.** *Given integers ℓ and q along with an n-vertex graph G, we can compute an ℓ-page stack or queue layout with page width at most q of G (if one exists) in time $n^{\mathcal{O}(q\cdot\ell)}$.*

The algorithm from Theorem 2 runs in polynomial time for constant ℓ and q. Furthermore, recall that deciding if a graph has a 2-page stack layout or 1-page queue layout is NP-complete [34, 46]. An ℓ-page 1-width stack or queue layout witnesses that G has cutwidth at most ℓ, which is in general NP-complete to decide [28]. This means that under well-established complexity assumptions, it is not possible to preserve XP-tractability if either of the two parameters is dropped.

5 Single-Exponential Algorithm for 1-Page Queue Layouts

In this section, we show that we can find a one-page queue layout, if it exists, in $2^{\mathcal{O}(n)}$ time, which improves on the trivial algorithm that exhaustively considers all $n!$ possible spine orders. Our algorithm exploits the equivalence between one-page queue layouts and arched leveled planar embeddings established by Heath and Rosenberg [34]. We first show that we can reduce the problem of deciding whether a graph admits a one-page queue layout to deciding whether at least one of $2^{\mathcal{O}(n)}$ *labeled* instances of Arched Leveled Planarity admits a solution. While Arched Leveled Planarity is NP-complete [34], our labeled instances can be solved in linear time thanks to a reduction to the well-studied problem Level Planarity [35, 36]. We will assume that G is connected (as each component can be treated separated) and, as G must be planar [34], we assume $|E| = \mathcal{O}(n)$.

We translate the definition of Heath and Rosenberg [34] into moderns terms as follows. Graph G has a *leveled planar embedding* if there exists a *level assignment* function $\gamma\colon V \to [h]$ that maps each $v \in V$ to one of $h \geq 1$ *levels*, and a straight-line planar drawing Γ of G such that each vertex v has y-coordinate $\gamma(v)$ and all edges are *proper*, i.e., for all $uv \in E$ we have $|\gamma(u) - \gamma(v)| = 1$. The latter property allows us to express the drawing as a collection of total orders $\prec_i$, one for each level $i \in [h]$, that specifies the left-to-right order of the vertices on level i [36]. For each level i, let s_i be the rightmost vertex on level i in Γ such that we have $s_i w \in E$ for some $w \in V$ with $\gamma(w) = i+1$, or if there is no such vertex, let s_i be the leftmost vertex on level i. *Arched leveled planar embeddings* extend leveled planar embeddings by also allowing non-monotone *arching* edges $t_i w$ between the leftmost vertex t_i on level i and vertices $w \in V$ with $\gamma(w) = i$ and $s_i \preceq_i w$ [34]. These edges can be drawn crossing-free below the lowest level; see Figure 7b. The problem Arched Leveled Planarity asks whether G has a level assignment that admits an arched leveled planar embedding. Heath



≺
a b c d e f g
(a)
(b)
g f e d c b a
$\gamma_{\mathcal{L}}(\cdot) = 4$
$\gamma_{\mathcal{L}}(\cdot) = 3$
$\gamma_{\mathcal{L}}(\cdot) = 2$
$\gamma_{\mathcal{L}}(\cdot) = 1$
(c)
g f e d c b a

Figure 7 **(a)** A one-page queue layout of G and **(b)** the corresponding arched leveled planar embedding obtained by the equivalence established by Heath and Rosenberg [34]. **(c)** The labeling induced by the embedding from **(b)**: Arcs labeled $\uparrow$ and $\circlearrowright$ are colored lilac and blue, respectively.

and Rosenberg [34] showed that the spine order of a one-page queue layout of a graph G induces a level assignment together with a vertex order on each level that together yield an arched leveled planar embedding of G and vice versa; see Figure 7.

We show that, when seeking a one-page queue layout of G, one can branch over the information needed to infer the level assignment for a corresponding arched leveled planar embedding. Consider an arbitrary orientation of the edge set E as an oriented arc set A and let $\delta\colon A \to \{\uparrow, \circlearrowright\}$ be a function that determines if an arc $uv \in A$ should be *ordinary* ($\uparrow$) or *arching* ($\circlearrowright$). The tuple $\mathcal{L} = (A, \delta)$ is called a *labeling* of G; see also Figure 7c. A level assignment γ is said to be *consistent* with $\mathcal{L}$ if and only if, for every $uv \in A$, it holds $\gamma(u) = \gamma(v) - 1$ if $\delta(uv) = \uparrow$ and $\gamma(u) = \gamma(v)$ if $\delta(uv) = \circlearrowright$. Similarly, an embedding Γ is consistent with $\mathcal{L}$ if its level assignment is consistent and we have $u \prec_{\gamma(u)} v$ for every $uv \in \delta^{-1}(\circlearrowright)$. Clearly, every arched leveled planar embedding Γ of G induces a labeling $(A_\Gamma, \delta_\Gamma)$ that is consistent with Γ. Conversely, we can reobtain Γ from this $(A_\Gamma, \delta_\Gamma)$ by fixing an arbitrary vertex to level 0, performing a BFS from v using the direction information in A_Γ to determine levels for all other vertices above and below, and potentially shifting all assigned levels such that their numbers span the same range as those in Γ. This is formalized by the following lemma.

▶ **Lemma 26** (★). *Let $\mathcal{L} = (A, \delta)$ be a labeling of G. In linear time we can compute a level assignment $\gamma_{\mathcal{L}}\colon V \to [h]$ that is consistent with the labeling or report that no such assignment exists. If there exists an arched leveled planar embedding Γ that induces $\mathcal{L}$, then there also exists one that induces $\mathcal{L}$ and uses the level assignment $\gamma_{\mathcal{L}}$.*

To now decide if a given graph G has an arched leveled planar embedding Γ, we can branch over all $\mathcal{O}(4^n) = 2^{\mathcal{O}(n)}$ different labelings $\mathcal{L} = (A, \delta)$ of G. In each branch, we use Lemma 26 to compute a level assignment $\gamma_{\mathcal{L}}$ or immediately reject the branch. Thus, it remains to check in each branch, i.e., for each level assignment, if G admits a corresponding arched leveled planar embedding. We do so using a linear-time reduction to the problem Level Planarity, which requires a level assignment as part of its input but does not allow arching edges. We emulate arching edges in Level Planarity as follows (see also Figure 8). First, we enclose the whole graph in a *frame*, which is a cycle spanning from below its lowest level to above its highest that will have a unique embedding (up to reflection) and for which we ensure that the remaining graph is drawn inside of it. Between each pair of levels $i, i+1$, we add a level $i + 0.5$ and subdivide all edges to ensure properness. We use l_i and r_i to refer to the two vertices of the frame cycle that are on level $i + 0.5$. Replace each arching arc $uv \in A$ with $\delta(uv) = \circlearrowright$ on level i with the edges ul_{i-1}, ul_i, and vr_i to G'; see Figure 8. Observe that this forces u to become the leftmost vertex on level i and v to be right of any

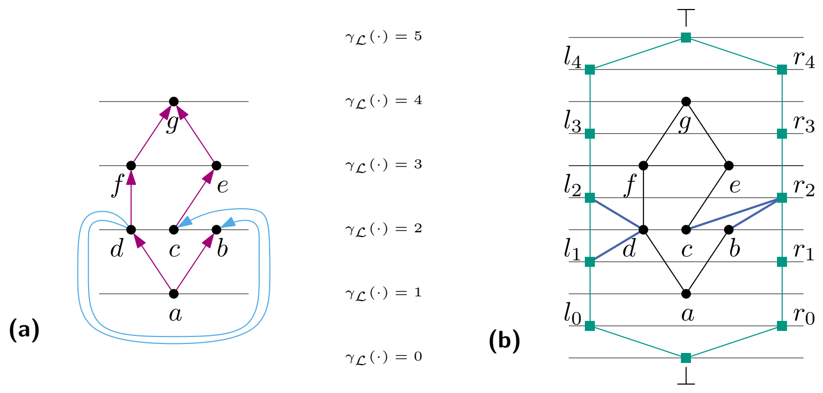

(a)
(b)
g
f
e
d
c
b
a
$\gamma_{\mathcal{L}}(\cdot) = 5$
$\gamma_{\mathcal{L}}(\cdot) = 4$
$\gamma_{\mathcal{L}}(\cdot) = 3$
$\gamma_{\mathcal{L}}(\cdot) = 2$
$\gamma_{\mathcal{L}}(\cdot) = 1$
$\gamma_{\mathcal{L}}(\cdot) = 0$
$\top$
$\bot$
l_4
l_3
l_2
l_1
l_0
r_4
r_3
r_2
r_1
r_0

Figure 8 (a) An instance of LABELED ARCHED LEVEL PLANARITY and **(b)** the obtained instance of LEVEL PLANARITY. We also visualize a drawing and color the frame and edges that simulate the arching edges from **(a)** in green and blue, respectively. Only relevant subdivision vertices are shown.

vertex adjacent to one on level $i + 1$. We provide a formal construction and show equivalence in the full version [14]. Using the known algorithm for testing LEVEL PLANARITY in linear time [35, 36], we finally obtain our third contribution.

▶ **Theorem 3** (★). *Given an n-vertex graph G, we can compute a 1-page queue layout of G (if one exists) in time $2^{\mathcal{O}(n)}$.*

6 Concluding Remarks

While our results improve the state of the art on computing linear layouts in several directions, they also highlight the prominent open questions in this area. In particular, Theorem 1 moves us closer to settling the long-standing open questions of whether treewidth or treedepth can be used to facilitate the computation of linear layouts [7, 8, 21, 25, 26]. At the same time, Theorems 2 and 3 yield the question of whether these classical problems can be solved in single-exponential time.

References

1 Akanksha Agrawal, Sergio Cabello, Michael Kaufmann, Saket Saurabh, Roohani Sharma, Yushi Uno, and Alexander Wolff. Eliminating Crossings in Ordered Graphs. In Hans L. Bodlaender, editor, *Proc. 19th Scandinavian Workshop on Algorithm Theory (SWAT'24)*, volume 294 of *LIPIcs*, pages 1:1–1:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, 2024. doi:10.4230/LIPICS.SWAT.2024.1.

2 Noga Alon and Vojtěch Rödl. Asymptotically tight bounds for some multicolored ramsey numbers.

3 Patrizio Angelini, Michael A. Bekos, Philipp Kindermann, and Tamara Mchedlidze. On mixed linear layouts of series-parallel graphs. *Theoretical Computer Science*, 936:129–138, 2022. doi:10.1016/J.TCS.2022.09.019.

4 Michael 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. doi:10.7155/JGAA.00479.

5 Michael A. Bekos, Martin Gronemann, and Chrysanthi N. Raftopoulou. An Improved Upper Bound on the Queue Number of Planar Graphs. *Algorithmica*, 85(2):544–562, 2023. doi:10.1007/S00453-022-01037-4.

6 Sujoy Bhore, Giordano Da Lozzo, Fabrizio Montecchiani, and Martin Nöllenburg. On the Upward Book Thickness Problem: Combinatorial and Complexity Results. In Helen C. Purchase and Ignaz Rutter, editors, *Proc. 29th International Symposium on Graph Drawing and Network Visualization (GD'21)*, volume 12868 of *Lecture Notes in Computer Science*, pages 242–256. Springer, 2021. doi:10.1007/978-3-030-92931-2_18.

7 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. doi:10.7155/JGAA.00526.

8 Sujoy Bhore, Robert Ganian, Fabrizio Montecchiani, and Martin Nöllenburg. Parameterized Algorithms for Queue Layouts. *Journal of Graph Algorithms and Applications*, 26(3):335–352, 2022. doi:10.7155/JGAA.00597.

9 Hans L. Bodlaender, Tesshu Hanaka, Yasuaki Kobayashi, Yusuke Kobayashi, Yoshio Okamoto, Yota Otachi, and Tom C. van der Zanden. Subgraph Isomorphism on Graph Classes that Exclude a Substructure. *Algorithmica*, 82(12):3566–3587, 2020. doi:10.1007/S00453-020-00737-Z.

10 Fan R. K. Chung, Frank Thomson Leighton, and Arnold L. Rosenberg. Embedding graphs in books: a layout problem with applications to VLSI design. *SIAM Journal on Algebraic Discrete Methods*, 8(1):33–58, 1987. doi:10.1137/0608002.

11 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015. doi:10.1007/978-3-319-21275-3.

12 Thomas Depian, Simon D. Fink, Robert Ganian, and Martin Nöllenburg. The Parameterized Complexity Of Extending Stack Layouts. In Stefan Felsner and Karsten Klein, editors, *Proc. 32nd International Symposium on Graph Drawing and Network Visualization (GD'24)*, volume 320 of *LIPIcs*, pages 12:1–12:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.GD.2024.12.

13 Thomas Depian, Simon D. Fink, Robert Ganian, and Martin Nöllenburg. The Peculiarities of Extending Queue Layouts. In *Proc. 51st International Workshop on Graph-Theoretic Concepts in Computer Science (WG'25)*, 2025. To appear.

14 Thomas Depian, Simon D. Fink, Robert Ganian, and Vaishali Surianarayanan. Linear Layouts Revisited: Stacks, Queues, and Exact Algorithms, 2025. Full Version. doi:10.48550/arXiv.2508.16319.

15 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate Texts in Mathematics*. Springer, 2012.

16 Pål Grønås Drange, Markus S. Dregi, and Pim van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016. doi:10.1007/S00453-016-0127-X.

17 Vida Dujmović, Gwenaël Joret, Piotr Micek, Pat Morin, Torsten Ueckerdt, and David R. Wood. Planar Graphs Have Bounded Queue-Number. *Journal of the ACM*, 67(4):22:1–22:38, 2020. doi:10.1145/3385731.

18 Vida Dujmovic, Pat Morin, and David R. Wood. Layout of Graphs with Bounded Tree-Width. *SIAM Journal on Computing*, 34(3):553–579, 2005. doi:10.1137/S0097539702416141.

19 Vida Dujmović and David R. Wood. On Linear Layouts of Graphs. *Discrete Mathematics & Theoretical Computer Science*, 6(2):339–358, 2004. doi:10.46298/DMTCS.317.

20 Vida Dujmovic and David R. Wood. Graph Treewidth and Geometric Thickness Parameters. *Discrete & Computational Geometry*, 37(4):641–670, 2007. doi:10.1007/S00454-007-1318-7.

21 Vida Dujmović and David R. Wood. On the Book Thickness of k-Trees. *Discrete Mathematics & Theoretical Computer Science*, 13(3):39–44, 2011. doi:10.46298/DMTCS.550.

22 Pavel Dvorák, Eduard Eiben, Robert Ganian, Dusan Knop, and Sebastian Ordyniak. Solving Integer Linear Programs with a Small Number of Global Variables and Constraints. In Carles Sierra, editor, *Proc 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*, pages 607–613, 2017. doi:10.24963/IJCAI.2017/85.

23 Pavel Dvořák, Eduard Eiben, Robert Ganian, Dušan Knop, and Sebastian Ordyniak. he complexity landscape of decompositional parameters for ILP: Programs with few global variables and constraints. *Artificial Intelligence*, 300:103561, 2021. `doi:10.1016/J.ARTINT.2021.103561`.

24 Robert Ganian, Fabian Klute, and Sebastian Ordyniak. On Structural Parameterizations of the Bounded-Degree Vertex Deletion Problem. *Algorithmica*, 83(1):297–336, 2021. `doi:10.1007/S00453-020-00758-8`.

25 Robert Ganian, Fabrizio Montecchiani, Martin Nöllenburg, and Meirav Zehavi. Parameterized Complexity in Graph Drawing (Dagstuhl Seminar 21293). *Dagstuhl Reports*, 11(6):82–123, 2021. `doi:10.4230/DAGREP.11.6.82`.

26 Robert Ganian, Haiko Müller, Sebastian Ordyniak, Giacomo Paesani, and Mateusz Rychlicki. A Tight Subexponential-Time Algorithm for Two-Page Book Embedding. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *Proc. 51st International Colloquium on Automata, Languages and Programming (ICALP'24)*, volume 297 of *LIPIcs*, pages 68:1–68:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. `doi:10.4230/LIPICS.ICALP.2024.68`.

27 Robert Ganian, Sebastian Ordyniak, and Maadapuzhi Sridharan Ramanujan. On Structural Parameterizations of the Edge Disjoint Paths Problem. *Algorithmica*, 83(6):1605–1637, 2021. `doi:10.1007/S00453-020-00795-3`.

28 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

29 Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theoretical Computer Science*, 918:60–76, 2022. `doi:10.1016/J.TCS.2022.03.021`.

30 Tesshu Hanaka, Michael Lampis, Manolis Vasilakis, and Kanae Yoshiwatari. Parameterized Vertex Integrity Revisited. In Rastislav Královic and Antonín Kucera, editors, *Proc. 49th International Symposium on Mathematical Foundations of Computer Science (MFCS'24)*, volume 306 of *LIPIcs*, pages 58:1–58:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. `doi:10.4230/LIPICS.MFCS.2024.58`.

31 Christian Haslinger and Peter F. Stadler. RNA Structures with Pseudo-knots: Graph-theoretical, Combinatorial, and Statistical Properties. *Bulletin of Mathematical Biology*, 61(3):437–467, 1999. `doi:10.1006/bulm.1998.0085`.

32 Lenwood S. Heath. Embedding outerplanar graphs in small books. *SIAM Journal on Algebraic Discrete Methods*, 8(2):198–218, 1987. `doi:10.1137/0608018`.

33 Lenwood S. Heath and Sriram V. Pemmaraju. Recognizing Leveled-Planar Dags in Linear Time. In Franz-Josef Brandenburg, editor, *Proc. 3rd International Symposium on Graph Drawing and Network Visualization (GD'95)*, volume 1027 of *Lecture Notes in Computer Science*, pages 300–311. Springer, 1996. `doi:10.1007/BFB0021813`.

34 Lenwood S. Heath and Arnold L. Rosenberg. Laying out Graphs Using Queues. *SIAM Journal on Computing*, 21(5):927–958, 1992. `doi:10.1137/0221055`.

35 Michael Jünger and Sebastian Leipert. Level Planar Embedding in Linear Time. In Jan Kratochvíl, editor, *Proc. 7th International Symposium on Graph Drawing and Network Visualization (GD'99)*, volume 1731 of *Lecture Notes in Computer Science*, pages 72–81. Springer, 1999. `doi:10.1007/3-540-46648-7_7`.

36 Michael Jünger and Sebastian Leipert. Level Planar Embedding in Linear Time. *Journal of Graph Algorithms and Applications*, 6(1):67–113, 2002. `doi:10.7155/JGAA.00045`.

37 Michael Jünger, Sebastian Leipert, and Petra Mutzel. Level Planarity Testing in Linear Time. In Sue Whitesides, editor, *Proc. 6th International Symposium on Graph Drawing and Network Visualization (GD'98)*, volume 1547 of *Lecture Notes in Computer Science*, pages 224–237. Springer, 1998. `doi:10.1007/3-540-37623-2_17`.

38 Julia Katheder, Michael Kaufmann, Sergey Pupyrev, and Torsten Ueckerdt. Transforming Stacks into Queues: Mixed and Separated Layouts of Graphs. In Olaf Beyersdorff, Michal Pilipczuk, Elaine Pimentel, and Kim Thang Nguyen, editors, *Proc. 42nd Symposium on Theoretical Aspects of Computer Science (STACS'25)*, volume 327 of *LIPIcs*, pages 56:1–56:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.STACS.2025.56.

39 Michael Lampis and Valia Mitsou. Fine-grained Meta-Theorems for Vertex Integrity. *Logical Methods in Computer Science*, 20, 2024. doi:10.46298/LMCS-20(4:18)2024.

40 Yunlong Liu, Jie Chen, Jingui Huang, and Jianxin Wang. On parameterized algorithms for fixed-order book thickness with respect to the pathwidth of the vertex ordering. *Theoretical Computer Science*, 873:16–24, 2021. doi:10.1016/J.TCS.2021.04.021.

41 Arnold L. Rosenberg. Book embeddings and wafer-scale integration. In *Proc. 17th Southeastern International Conference on Combinatorics, Graph Theory, and Computing*, volume 54, pages 217–224, 1986.

42 James B. Saxe. Dynamic-Programming Algorithms for Recognizing Small-Bandwidth Graphs in Polynomial Time. *SIAM Journal on Algebraic Discrete Methods*, 1(4):363–369, 1980. doi:10.1137/0601042.

43 Elena Stöhr. A Trade-off between Page Number and Page Width of Book Embeddings of Graphs. *Information and Computation*, 79(2):155–162, 1988. doi:10.1016/0890-5401(88)90036-3.

44 Elena Stöhr. The pagewidth of trivalent planar graphs. *Discrete Mathematics*, 89(1):43–49, 1991. doi:10.1016/0012-365X(91)90398-L.

45 Veit Wiechert. On the queue-number of graphs with bounded tree-width. *Electron. J. Comb.*, 24(1):1, 2017. doi:10.37236/6429.

46 Mihalis Yannakakis. Embedding Planar Graphs in Four Pages. *Journal of Computer and System Sciences*, 38(1):36–67, 1989. doi:10.1016/0022-0000(89)90032-9.