Parameterized Algorithms for Queue Layouts

Sujoy Bhore\textsuperscript{1} Robert Ganian\textsuperscript{2} Fabrizio Montecchiani\textsuperscript{3} Martin Nöllenburg\textsuperscript{2}

\textsuperscript{1}Computer Science Department, Université libre de Bruxelles (ULB), Bruxelles, Belgium
\textsuperscript{2}Algorithms and Complexity Group, TU Wien, Vienna, Austria
\textsuperscript{3}Engineering Department, University of Perugia, Perugia, Italy

Abstract. An \textit{h}-queue layout of a graph $G$ consists of a linear order of its vertices and a partition of its edges into $h$ sets, called queues, such that no two independent edges of the same queue nest. The minimum $h$ such that $G$ admits an \textit{h}-queue layout is the \textit{queue number} of $G$. We present two fixed-parameter tractable algorithms that exploit structural properties of graphs to compute optimal queue layouts. As our first result, we show that deciding whether a graph $G$ has queue number 1 and computing a corresponding layout is fixed-parameter tractable when parameterized by the treedepth of $G$. Our second result then uses a more restrictive parameter, the vertex cover number, to solve the problem for arbitrary $h$.

1 Introduction

An \textit{h}-queue layout of a graph $G$ is a linear layout of $G$ consisting of a linear order of its vertices and a partition of its edges into $h$ sets, called queues, such that no two independent edges of the same queue nest \cite{32}; see Fig. 1 for an illustration. The \textit{queue number} $\text{qn}(G)$ of a graph $G$ is the minimum number of queues in any queue layout of $G$. While such linear layouts represent an abstraction of various problems such as, for instance, sorting and scheduling \cite{6,40}, they also play a central role in three-dimensional graph drawing. It is known that a graph class has bounded queue...
number if and only if every graph in this class has a three-dimensional crossing-free straight-line grid drawing in linear volume \([15, 22]\). We refer the reader to \([24, 37]\) for further references and applications. Moreover, it is worth recalling that stack layouts \([36, 42]\) (or book embeddings \([48, 44]\)), which allow nesting edges but forbid edge crossings, form the “dual” concept of queue layouts.

A rich body of literature is concerned with the study of upper bounds for the queue number of several planar and non-planar graph families (see, e.g., \([1, 4, 5, 9, 14, 19, 21, 22, 31, 41]\) and also \([23]\) for additional references). For instance, a graph of treewidth \(w\) has queue number at most \(O(2^w)\) \([41]\), while every proper minor-closed class of graphs (including planar graphs) has constant queue number \([21]\).

Of particular interest to us is the corresponding recognition problem, which we denote by \textsc{Queue Number}: Given a graph \(G\) and a positive integer \(h\), decide whether \(G\) admits an \(h\)-queue layout. In 1992, in a seminal paper, Heath and Rosenberg proved that 1-\textsc{Queue Number}, i.e., the restriction of \textsc{Queue Number} to instances with \(h = 1\), is \textsc{NP}-complete \([32]\). In particular, they characterized the graphs that admit queue layouts with only one queue as the arched leveled-planar graphs, and showed that the recognition of these graphs is \textsc{NP}-complete \([32]\).

Since \textsc{Queue Number} is \textsc{NP}-complete even for a single queue, it is natural to ask under which conditions the problem can be solved efficiently. For instance, it is known that if the linear order of the vertices is given (and the aim is thus to simply partition the edges of the graph into \(h\) queues), then the problem becomes solvable in polynomial time \([31]\). We follow up on recent work made for the stack number \([7]\) and initiate the study of the parameterized complexity of \textsc{Queue Number} by asking under which parameterizations the problem is fixed-parameter tractable. In other words, we are interested in whether (1-)\textsc{Queue Number} can be solved in time \(f(k) \cdot n^{O(1)}\) for some computable function \(f\) of the considered structural parameter \(k\) of the \(n\)-vertex input graph \(G\). Parameterized complexity is a modern algorithmic paradigm that allows us to obtain a more fine-grained understanding of the complexity of difficult problems, and it has recently gained increasing attention in the graph drawing community; see the recent Dagstuhl seminar for more information about the paradigm’s limitations and applicability \([27]\). Parameterized complexity has been successfully applied on graph drawing problems related to 1-planarity \([3, 26]\), crossing minimization \([33]\), layered graph drawing \([20]\), linear layouts \([7, 9]\), orthogonal planarity \([16]\), upward planarity \([10]\), and others.

As our main result, we show 1-\textsc{Queue Number} is fixed-parameter tractable parameterized by the treedepth of the input graph (Section 3). We remark that treedepth is a fundamental graph parameter with close ties to the theory of graph sparsity (see, e.g., \([35]\)). The main technique used by the algorithm is iterative pruning, where we recursively identify irrelevant parts of the input and remove these until we obtain a bounded-size equivalent instance (a \textit{kernel}) solvable by
brute force. While the iterative pruning technique has already been used in a few other algorithms that exploit treedepth [28–30], the unique challenge here lay in establishing that the removal of seemingly irrelevant parts of the graph cannot change NO-instances to YES-instances. The proof of this claim, formalized in Lemma 1, uses a new type of block decomposition of 1-queue layouts.

For our second result, we turn to the general QUEUE NUMBER problem. Here, we establish fixed-parameter tractability when parameterized by a larger parameter, namely the vertex cover number (Section 4). This result is also achieved by kernelization and forms a natural counterpart to the recently established fixed-parameter tractability of computing the stack number under the same parameterization [7], see also recent work on upward book thickness [9], although the technical arguments and steps of the proof differ due to the specific properties of queue layouts.

2 Preliminaries

We adopt standard notation and terminology from graph theory [17]. We can assume that our input graphs are connected, as the queue number of a graph is the maximum queue number over all its connected components. Given a graph $G = (V, E)$ and a vertex $v \in V$, let $N(v)$ be the set of neighbors of $v$ in $G$. Also, for $r \in \mathbb{N}$, we denote by $[r]$ the set $\{1, \ldots, r\}$. An $h$-queue layout of $G$ is a pair $\langle \prec, \sigma \rangle$, where $\prec$ is a linear order of $V$, and $\sigma : E \rightarrow [h]$ is a function that maps each edge of $E$ to one of $h$ sets, called queues. In an $h$-queue layout $\langle \prec, \sigma \rangle$ of $G$, it is required that no two independent edges in the same queue nest, that is, for no pair of edges $uv, wx \in E$ with four distinct end-vertices and $\sigma(uv) = \sigma(wx)$, the vertices are ordered as $u \prec w \prec x \prec v$. Given two distinct vertices $u$ and $v$ of $G$, $u$ is to the left of $v$ if $u \prec v$, else $u$ is to the right of $v$. Note that a 1-queue layout of $G$ is simply defined by a linear order $\prec$ of $V$ and $\sigma \equiv 1$.

We assume familiarity with basic notions in parameterized complexity [12, 18]. We consider two graph parameters for our algorithms: treedepth and vertex cover number.

2.1 Treedepth

Treedepth is a parameter closely related to treewidth, and the structure of graphs of bounded treedepth is well understood [35]. A useful way of thinking about graphs of bounded treedepth is that they are (sparse) graphs with no long paths. We formalize a few notions needed to define treedepth, see also Fig. 2 for an illustration. A rooted forest $F$ is a disjoint union of rooted trees. For a vertex $x$ in a tree $T$ of $F$, the height (or depth) of $x$ in $F$ is the number of vertices in the path from the root of $T$ to $x$. The height of a rooted forest is the maximum height of a vertex of the forest. Let $V(T)$ be the vertex set of any tree $T \in F$.

**Definition 1 (Treedepth)** Let the closure of a rooted forest $F$ be the graph $\text{clos}(F) = (V_c, E_c)$ with the vertex set $V_c = \bigcup_{T \in F} V(T)$ and the edge set $E_c = \{xy \mid x$ is an ancestor of $y$ in some $T \in F\}$. A treedepth decomposition of a graph $G$ is a rooted forest $F$ such that $G \subseteq \text{clos}(F)$. The treedepth $\text{td}(G)$ of a graph $G$ is the minimum height of any treedepth decomposition of $G$.

An optimal treedepth decomposition can be computed by an FPT algorithm.

**Proposition 1** ([39]) Given an $n$-vertex graph $G$ and an integer $k$, it is possible to decide whether $G$ has treedepth at most $k$, and if so, to compute a treedepth decomposition of $G$ of height at most $k$ in time $2^{O(k^2)} \cdot n$.

**Proposition 2** ([35]) Let $G$ be a graph and $\text{td}(G) \leq k$. Then $G$ has no path of length $2^k$. 


2.2 Vertex cover number

A vertex cover $C$ of a graph $G = (V, E)$ is a subset $C \subseteq V$ such that each edge in $E$ has at least one incident vertex in $C$. The vertex cover number of $G$, denoted by $\tau(G)$, is the size of a minimum vertex cover of $G$. Observe that $td(G) \leq \tau(G) + 1$: it suffices to build $F$ as a single path with vertex set $C$ and with the leaves $V \setminus C$ all placed below the last vertex of this path. Computing an optimal vertex cover of $G$ is FPT.

**Proposition 3** ([11]) Given an $n$-vertex graph $G$ and a constant $\tau$, it is possible to decide whether $G$ has vertex cover number at most $\tau$, and if so, to compute a vertex cover $C$ of size $\tau$ of $G$ in time $O(2^{\tau + \tau \cdot n})$.

3 Parameterization by Treedepth

In this section, we establish our main result: the fixed-parameter tractability of 1-Queue Number parameterized by treedepth. We formalize the statement below.

**Theorem 1** Let $G$ be a graph with $n$ vertices and constant treedepth $k$. We can decide in $O(n)$ time whether $G$ has queue number one, and, if this is the case, we can also output a 1-queue layout of $G$ in the same time.

3.1 Algorithm Description

Since we assume $G$ to be connected, any treedepth decomposition of $G$ consists of a single tree $T$. Now, suppose that a treedepth decomposition $T$ of $G$ of depth $k$ is given. For a vertex $t$ of $T$, let $P_t$ be the set of ancestors of $t$ including $t$, let $A_t$ be the set of connected components of $G - P_t$ that contain a child of $t$, and $m_t$ be the maximum number of vertices in a component in $A_t$; see also Fig. 2(b). Notice that $|A_t|$ is precisely the number of children of $t$ in $T$.

**Observation 1** For every component $C \in A_t$ and for every vertex $v \in C$, it holds that $N(v) \subseteq C \cup P_t$. Thus, $|C \cup P_t| \leq m_t + k$. 

Figure 2: (a) A graph $G$ and (b) a treedepth decomposition $T$ of $G$ of height 4, in which $T$ has light red edges. In particular, $P_2 = \{1, 2\}$, $A_2 = \{C_1, C_2, C_3\}$, and $m_2 = 3$. The tree is highlighted in red.
Now, we define the following equivalence over components in $A_t$. Components $C, D \in A_t$ satisfy $C \sim D$ if and only if there exists a bijective renaming function $\eta_{C,D} : C \to D$ over (the vertices of) $C, D$ such that each vertex $c_i \in C$ has a counterpart $\eta_{C,D}(c_i) = d_i \in D$ that satisfies: (i) $N(c_i) \cap P_i = N(d_i) \cap P_i$ and (ii) $c_i$ is adjacent to $c_j \in C$ if and only if $d_i$ is adjacent to its counterpart $d_j$. When $C, D$ are clear from the context, we may drop the subscript $\eta$ for brevity. For an example of two equivalent components, see the subtrees rooted at vertices 4 and 4′ in Fig. 2(b).

By Observation 1, the number of equivalence classes of $\sim$ is upper-bounded by the number of possible graphs on $k + m_t$ vertices, which is at most $2^{(k + m_t)^2}$. The next observation allows us to propagate the bounds formalized by the notation above from children towards the root.

**Observation 2** If for a vertex $t$ of $T$ there exist integers $a, b$ such that each child $q$ of $t$ satisfies $|A_q| \leq a$ and $m_q \leq b$, then $m_t \leq (a \cdot b) + 1$.

The main component of our treedepth algorithm is Lemma 1, stated below. Intuitively, applying Lemma 1 bottom-up on $T$ (together with Observation 2) allows us to iteratively remove subtrees from $T$ while preserving the (non-)existence of a hypothetical solution—in particular, we will be able to prune subtrees of parents with a very large number of children until we reach an equivalent instance where each vertex has a bounded number of children. To formalize the meaning of “very large”, we define the following function for $k, i \geq 2$ (recalling that $k$ is the depth of $T$):

$$
\#children(k, i) = ((\lceil (\theta(k) + 1)^{\text{size}(k,i)^2} \rceil + 1) \cdot (\text{size}(k,i) + 1)!) \cdot 2^{(k + \text{size}(k,i))^2},
$$

where $\text{size}(k, i)$ is a recursively defined function that captures the size bound given by Observation 2 as follows:

- $\text{size}(k, i) = (\text{size}(k, i - 1) \cdot \#children(k, i - 1)) + 1$ for $i \geq 2$, and
- $\text{size}(k, 1) = \#children(k, 1) = 0$.

As an example, we note that while $\text{size}(k, 2) = 1$, the value of $\#children(k, 2)$ is already in $k^{\Theta(k)}$ and both functions experience an exponential jump with each increase of $i$ from there on. Intuitively, the precise values of the functions are set to guarantee that if one has successfully completed pruning for subtrees on lower levels of the treedepth decomposition, and if at the same time the number of children of a vertex at depth $i$ is greater than $\#children(k, i)$, we will find an equivalence class at level $i$ that is sufficiently large to guarantee the correctness of the pruning step. This intuition is formalized in the aforementioned Lemma 1 (and readers are invited to compare the definitions of these functions with the way they are used in its proof):

**Lemma 1** Assume $G$ has a vertex $t$ at depth $i$ in $T$ with the property that $|A_t| \geq \#children(k, i)$, but $m_t \leq \text{size}(k, i)$ and every descendant $q$ of $t$ in $T$ satisfies that $|A_q| \leq \#children(k, i - 1)$. Then there exists a component $B$ of $A_t$ such that $G - B$ has queue number one if and only if $G$ has queue number one. Moreover, $B$ can be computed in time $O(\text{size}(k, i)! \cdot \#children(k, i)^2)$.

The proof of the lemma is deferred to Section 3.2. Before proceeding, we show how Lemma 1 is used to obtain Theorem 1.

**Proof:** [of Theorem 1] We start by applying Proposition 1 to compute a treedepth decomposition $T$ of $G$ of depth at most $k$. Consider now vertices at depth $k - 1$ in $T$, i.e., vertices whose children are all leaves in $T$, and set $i = 2$. Observe that every vertex $v$ at this depth satisfies
Let \( B \) queue number. On the other hand, assume there is a 1-queue layout of \( G \). Recall that we are, at this stage, proceeding under the assumption that every hypothetical solution must contain two components which behave “in the same way” (as will become clear later). Moreover, this equivalence class can be computed in time at most \( 1 \) components in the equivalence class is that this will allow us to argue that \( A \), must exist an equivalence class, denoted \( A_t \), containing at least \((2^{k+1}+1)\text{size}(k,i)^2+1\)·(\text{size}(k,i)+k)! \cdot 2^{k+\text{size}(k,i)^2} = \#children(k,i)\)

and the number of equivalence classes of \( \sim \) is upper-bounded by \( 2^{k+\text{size}(k,i)^2} \), there must exist an equivalence class, denoted \( A_t \), containing at least \((2^{k+1}+1)\text{size}(k,i)^2+1\)·(\text{size}(k,i)+k)! connected components in \( A_t \) which are pairwise equivalent w.r.t. \( \sim \). The reason we need this many components in the equivalence class is that this will allow us to argue that every hypothetical solution must contain two components which behave “in the same way” (as will become clear later). Moreover, this equivalence class can be computed in time at most \( \text{size}(k,i)! \cdot \#children(k,i)^2 \) by simply brute-forcing over all potential renaming functions \( \eta \) between arbitrarily chosen \( \#children(k,i) \)-many components in \( A_t \) to construct the set of all equivalence classes of these components. Let \( B \) be an arbitrarily selected component in \( A_t^* \). First, observe that if \( G \) is a YES-instance then so is \( G - B \), as deleting vertices and edges cannot increase the queue number. On the other hand, assume there is a 1-queue layout of \( G - B \) with linear order \( \prec \). Our aim for the rest of the proof is to obtain a linear order \( \prec \) of \( G \) that extends \( \prec \) and yields a valid 1-queue layout of \( G \).

### 3.2 Proof of Lemma 1

Since we have

\[
|A_t| \leq \left((2^{k+1}+1)\text{size}(k,i)^2+1\right) \cdot \left(\text{size}(k,i)+k\right)! \cdot 2^{k+\text{size}(k,i)^2} = \#children(k,i)\]

and recall from Observation 2 that every vertex \( v \) satisfies \( d_v < \text{size}(k,i) \). In particular, assume that for some depth \( d_v \leq \text{size}(k,i) \) and \( m_v \leq \text{size}(k,i) \).

Once the above procedure terminates for the last time, the root \( r \) of \( T \) satisfies \( |A_r| < \#children(k,k) \) and \( m_r \leq \text{size}(k,k) \). At that point, we have a kernel \( G' \) [12, 18]—an equivalent graph that has size bounded by a function of \( k \), notably by \( f(k) = \#children(k,k) \cdot \text{size}(k,k) + 1 \).

To prove Theorem 1, it suffices to decide whether \( G' \) admits a 1-queue layout by a brute-force algorithm that runs in time \( \mathcal{O}(f(k)! \cdot f(k)^2) \). Since Lemma 1 is applied \( \mathcal{O}(n) \) times and the runtime of the associated algorithm is \( \mathcal{O}(\text{size}(k,k) \cdot \#children(k,k)^2) \), the total runtime is upper-bounded by a function of \( k \) times \( n \).

Finally, we note that while it would be possible to provide a term upper-bounding the dependency on \( k \) of the running time of Lemma 1, it is clear that such a term must necessarily be non-elementary—indeed, the recursive definition of the two functions \( \#children(k,k) \) and \( \text{size}(k,k) \) results in a tower of exponents of height \( k \).

A Refined Equivalence. Recall that we are, at this stage, proceeding under the assumption that there exists a 1-queue layout of \( G - B \) with linear order \( \prec \). Let \( \equiv \prec \) be an equivalence over components in \( A_t^* \) which takes this hypothetical order \( \prec \) into account and is defined as follows.
Figure 3: Two delimiting components $C_1$ and $C_2$ (blue and red), with two counterpart (and hence interleaving) edges labeled. Notice that no two counterpart edge pairs are separate.

For two components $C, D \in A^\sim_k$, $C \equiv \prec D$ if and only if the following holds: the linear order $\prec$ restricted to $P_t \cup \eta_{C,D}(C)$ is the same as $\prec$ restricted to $P_t \cup C$. In other words, $\equiv \prec$ is a refinement of $\sim \prec$ restricted to $A^\sim_k$ which groups components based on the order in which their vertices appear (also taking into account which subinterval they appear in w.r.t. $P_t$). Note that $\equiv \prec$ has at most $(m_t + k)! \leq (\text{size}(k,i) + k)!$ many equivalence classes, since $|P_t| \leq k$; hence, by the virtue of $A^\sim_k$ having size at least $((2^{k+1} + 1) \text{size}(k,i)^2 + 1) \cdot (\text{size}(k,i) + k)!$, there must exist an equivalence class $U$ of $\equiv \prec$ containing at least $(2^{k+1} + 1) \text{size}(k,i)^2 + 1$ components of $A^\sim_k$.

We adopt the following terminology for $U$: we will denote the components in $U$ as $C_1, C_2, \ldots, C_u$, where $u = |U|$, we will identify the vertices in a component $C_i$ by using the lower index $i$, and for each such vertex $v$, say $v = v_i \in C_i$, use $v_j$ to denote its counterpart $\eta_{C_i,C_j}(v_i)$.

**Identifying Delimiting Components.** Consider two adjacent vertices $v_i, w_i$ in $C_i$. We say that component $C_j$ is $vw$-separate from $C_i$ if edges $v_iw_i$ and $v_jw_j$ neither nest nor cross each other. On the other hand, $C_j$ is $vw$-interleaving (respectively, $vw$-nesting) with $C_i$ if $v_iw_i$ and $v_jw_j$ cross each other (respectively, if one of $v_iw_i$ and $v_jw_j$ nests the other). By the definition of $\equiv \prec$ and $U$, these three cases are exhaustive. Moreover, if $v_iw_i$ is an edge then so is $v_jw_j$ and hence $C_j$ cannot be $vw$-nesting with $C_i$.

Our next aim will be to find two components—we will call them *delimiting components*—that are not $vw$-separate for any edge $vw$, see, e.g., Fig. 3. To this end, for some two adjacent vertices $v_i, w_i$ of $C_i$, denote by $D_t$ the component whose counterpart to $v_i$ (say $v_1$) is placed leftmost in $\prec$ among all components in $U$. We now define a sequence of components as follows: $D_t$ is the unique component that is (i) $vw$-separate from $D_{t-1}$ and (ii) whose vertex $v_t$ is placed to the right of $v_{t-1}$, and (iii) $v_t$ is placed leftmost among all components satisfying properties (i) and (ii). Let $d$ be the maximum integer such that $D_d$ exists.

**Lemma 2** $d \leq 2^{k+1} + 1$.

**Proof:** Consider, for a contradiction, that there exists a component $D_t$ such that $\ell > 2^k$ and $\ell < d - 2^k$, i.e., that there is a sequence of at least $2^k$ pairwise $vw$-separated components to the left as well as to the right of $D_t$. By the connectivity of $G$, there must be a path from $v$ to some vertex in $P_t$, say $p$. However, by the definition of $\equiv \prec$ every vertex in $P_t$ lies either to the left of $v_1$ or to the right of $w_d$, and hence a path from $v$ to $p$ would need to pass through a sequence of $2^k$ edges forming disjoint intervals in the linear order $\prec$. Since nestings are not allowed, such a path must have at least one vertex inside each of these intervals, and hence its length is at least $2^k$, which contradicts Proposition 2. \qed
Moreover, each component \( C_q \) in \( U \) can be uniquely assigned to one component \( D_\ell \) as defined above (w.r.t. the chosen edge \( vw \)) as follows: If \( C_q = D_\ell \) for some \( \ell \), then \( C_q \) is assigned to itself; otherwise, \( D_\ell \) is the component whose vertex \( v_\ell \) is to the left of and simultaneously closest to the corresponding vertex \( v_q \) in \( C_q \) among all components \( D_1, \ldots, D_\ell \).

**Lemma 3** Let \( C_q \) and \( C_p \) be two components assigned to the same component \( D_\ell \) w.r.t. the edge \( vw \). Then \( C_q \) and \( C_p \) are \( vw \)-interleaving.

**Proof:** Assume without loss of generality that some vertex \( v \prec w \) in \( C_q \). Without loss of generality and recalling Lemma 4, we will hereinafter assume that every vertex \( x \in P \) first edge on \( \varnothing \). Because edges cannot nest on the same queue, this implies that the counterparts \( v \) and \( w \) must be placed to the left of \( w \). Hence \( C_q \) and \( C_p \) cannot be \( vw \)-separate, and the observation follows by recalling that \( C_q \) and \( C_p \) cannot be \( vw \)-nesting either. \( \square \)

We are now ready to construct our delimiting components. Recall that at this point, \(|U| \geq (2^{(k+1)} + 1) \cdot \text{size}(k,i)^2 + 1 \) while the maximum number of edges inside a component in \( U \) is upper-bounded by \( n^2 \cdot \text{size}(k,i)^2 \). Hence by the pigeon-hole principle and by applying the bound provided in Lemma 2 for each edge inside the components of \( U \), there must exist two components in \( U \), say \( C_x \) and \( C_y \), that are assigned to the same component \( D_\ell \) for each edge \( vw \). By Lemma 3 it now follows that they are \( vw \)-interleaving for every edge \( vw \).

**Using Delimiting Components.** Before we use \( C_x \) and \( C_y \) to insert the component \( B \) of \( A_t \) as required by Lemma 1, we can show that the way they interleave with each other is “consistent” in \( \prec \).

**Lemma 4** Assume without loss of generality that some vertex \( v_x \) is to the left of \( v_y \). Then for each vertex \( w \) it holds that \( w \) is to the left of \( w \).

**Proof:** Consider for a contradiction that there is a vertex \( w \) to the right of \( w \). Consider a \( v_x-w \) path \( P_x \) in the subgraph of \( G \) induced on the vertices of \( C_x \), and let \( P_y \) be the \( v_y-w_y \) path in the subgraph of \( G \) induced on the vertices of \( C_y \) consisting of the counterparts of \( P_x \). Let \( a_xb_x \) be the first edge on \( P_x \) such that \( a_x \) is placed to the left of \( a_y \) but \( b_x \) is placed to the right of \( b_y \). Then the edges \( a_xb_x \) and \( a_yb_y \) would be nesting, contradicting the correctness of \( \prec \). \( \square \)

We remark that it is not the case that \( C_x \) must be \( vw \)-interleaving with \( C_y \) if \( vw \) is not an edge—this is, in fact, a major complication that we will need to overcome to complete the proof.

Without loss of generality and recalling Lemma 4, we will hereinafter assume that every vertex \( v_x \in C_x \) is placed to the left of its counterpart \( v_y \in C_y \). The following definition allows us to partition the vertices of \( C_x \) into subsequences that should not be interleaved with vertices of \( B \).

**Definition 2 (Block)** A block \( L = \{v^1_x, v^2_x, \ldots, v^h_x\} \) of \( C_x \) is a maximal set of vertices of \( C_x \) such that: (1) there is no vertex \( v^i_y \) (the counterpart in \( C_y \) of \( v^i_x \)), with \( 1 \leq i \leq h \), between two vertices of \( L \) in \( \prec \); and (2) there are no two vertices of \( L \) such that one has a neighbor to its left and one has a neighbor to its right.

We observe that, as an immediate consequence of Definition 2, no two vertices of \( L \) are adjacent (an edge \( uv \) in \( L \) would imply that \( u \) has a neighbor to its right and \( v \) has a neighbor to its left, or vice versa).
For each block \( L = \{v_{1}^{x}, v_{2}^{x}, \ldots, v_{h}^{x}\} \) of \( C_{x} \), there is a corresponding set of vertices \( \{v_{B}^{1}, v_{B}^{2}, \ldots, v_{B}^{h}\} \) of \( B \), i.e., the set containing the counterparts of \( L \) in \( B \). We will obtain a linear order of \( G \) by processing the blocks of \( C_{x} \) one by one as encountered in a left-to-right sweep of \( \prec \), and for each block \( L \), we will extend \( \prec \) by suitably inserting the corresponding vertices of \( B \).

Consider the \( i \)-th encountered block \( L_{i} = \{v_{i,1}^{x}, v_{i,2}^{x}, \ldots, v_{i,\ell_{i}}^{x}\} \) of \( C_{x} \), refer to Fig. 4 for an illustration. Note that, because \( C_{x} \) and \( C_{y} \) are equivalent components, it holds \( v_{i,1}^{x} \prec v_{i,2}^{x} \prec \cdots \prec v_{i,\ell_{i}}^{x} \) (even though such vertices might not be consecutive). Also, let \( v_{i} \) be the first vertex to the left of \( v_{i,1}^{x} \) in \( \prec \) (possibly \( v_{i} = v_{i,\ell_{i}}^{x} \)). We insert all vertices in the corresponding block \( B_{i} \) of \( B \) such that:

\[
\begin{align*}
v_{i} \prec v_{i,1}^{x} \prec v_{i,2}^{x} \prec \cdots \prec v_{i,\ell_{i}}^{x} \prec v_{i,1}^{y}.
\end{align*}
\]

After processing the last block of \( C_{x} \), we know that all vertices of \( C_{x} \) have been considered and hence all vertices of \( B \) have been reinserted, that is, we extended \( \prec \) to a linear order \( \prec' \) of the whole graph \( G \). The next observation immediately follows by the procedure described above.

**Observation 3** For every vertex \( v_{x} \), it holds that \( v_{x} \prec' v_{B} \prec' v_{y} \).

We now establish the correctness of \( \prec' \), completing the proof of Lemma 1.

**Lemma 5** The linear order \( \prec' \) yields a valid 1-queue layout of \( G \).

**Proof:** To prove the statement, we argue that no two edges of \( G \) nest in the 1-queue layout defined by \( \prec' \). We recall that \( \prec' \) extends \( \prec \), hence we do not need to argue about pairs of edges in \( G - B \). Moreover, by construction, \( \prec' \) restricted to \( C_{x} \) is the same as \( \prec \) restricted to \( B \) (up to the renaming function \( \eta \)). Consequently, no two edges having both endpoints in \( B \) can nest. To complete the proof, it suffices to consider the two cases of an edge having only one endpoint or both endpoints in \( B \) (i.e., the "newly added" edges), and show that no such edge can be involved in any nesting.
We first consider any edge $v_Bw$ for $w \in P_t$ and $v_B \in B$, and assume $v_B \prec w$ (else the argument is symmetric). Suppose, for a contradiction, that $v_Bw$ nests another edge $ab$. Recall that since $C_x$ and $B$ are equivalent components, if $v_B$ is to the left of $w$, the same holds for $v_x$. By Observation 3, we know $v_x \prec v_B \prec w$, which implies that $ab$ is nested by $v_xw$ as well, a contradiction with the correctness of $\prec$. Similarly, if $v_Bw$ is nested by an edge $ab$, then we know $v_B \prec v_y \prec w$, which implies that $ab$ nests $v_yw$ as well, again a contradiction.

We now consider any edge $v_Bw_B$, with $v_B \prec w_B$. We further distinguish whether, for a contradiction, $v_Bw_B$ nests an edge $ab$ or is nested by an edge $ab$.

- Assume $v_Bw_B$ nests an edge $ab$. Since Definition 2 ensures that a block cannot contain a pair of adjacent vertices, we know that $v_x$ and $w_x$ belong to different blocks, say $L_i$ and $L_j$ (with $i < j$) respectively. Therefore, we can rename the vertices as $v_x = v_x^{i,i'}$ and $w_x = v_x^{j,j'}$, and similarly $v_B = v_B^{i,i'}$ and $w_B = v_B^{j,j'}$; refer to Fig. 5(a) for an illustration. By Observation 3, it holds $v_x^{i,i'} \prec v_B^{i,i'} \prec v_B^{j,j'}$ and $v_B^{j,j'} \prec v_B^{j,j'}$. Moreover, the correctness of $\prec$ implies that $v_B^{i,i'} \prec a \prec v_B^{i,i'}$ (since $v_B^{i,i'}$ cannot nest $ab$) and $v_B^{j,j'} \prec b \prec v_B^{j,j'}$ (since $v_B^{i,i'}$ cannot nest $ab$). Because $a$ is between $v_B^{i,i'}$ and $v_B^{j,j'}$, either there exists another vertex $v_B^{y,y'}$ (the counterpart to the first vertex in block $L_i$, where possibly $v_B^{y,y'} = a$) such that $v_B^{i,i'} \prec v_B^{y,y'} \prec a \prec v_B^{i,i'}$.

Suppose first $a \neq v_B^{i,i'}$ and $b \neq v_B^{i,i'}$. Observe that $v_B^{i,i'}$ has at least one neighbor in $C_x$ (because $C_x$ is connected), and that $v_B^{j,j'}$ is to the right of $v_B^{i,i'}$, hence, by Definition 2, $v_B^{i,i'}$ also has a neighbor to its right, say $v_B^{j,j'}$. Because no two edges nest in $\prec$, it must be: (i) $v_B^{i,i'} \prec v_B^{j,j'}$, (ii) $v_B^{j,j'} \prec b$, and (iii) $v_B^{j,j'} \prec b$ (possibly $v_B^{j,j'} = b$). Altogether, this implies that $v_B^{j,j'}$ and $v_B^{j,j'}$ are in the same block (i.e., $l = j$) and hence $v_B^{i,i'} \prec v_B^{j,j'} \prec b$, which contradicts $b \prec v_B^{j,j'}$. If instead $a = v_B^{i,i'}$ and $a = v_B^{i,i'}$, then $b$ is either a vertex of $C_y$ or a vertex of $P_t$. If $b \in C_y$, the argument is similar, as we can set $b = v_B^{i,i'}$ and observe that $v_B^{j,j'}$ should be to the left of $v_B^{i,i'}$, see Fig. 5(b). If $b \in P_t$, we would have $v_B^{j,j'} \prec b \prec v_B^{j,j'}$, which contradicts the fact that $C_x$ and $C_y$ are equivalent components, see Fig. 5(c).

- Assume now that $v_Bw_B$ is nested by an edge $ab$. Again we can rename the vertices as $v_x = v_x^{i,i'}$ and $w_x = v_x^{j,j'}$, and similarly $v_B = v_B^{i,i'}$ and $w_B = v_B^{j,j'}$. By the position of $b$ we can deduce either that $b = v_B^{i,i'}$ (possibly $j' = 1$) or that edge $v_B^{i,i'}v_B^{j,j'}$ exists. In the latter case either $v_B^{i,i'}v_B^{j,j'}$ is also nested by $ab$ or $v_B^{i,i'} \prec a$, and in both cases we obtain a contradiction; refer to Fig. 6(a) for an illustration. In the former case, we should again distinguish whether $a \in C_y$ or $a \in P_t$. If $a \in C_y$, it should be $v_B^{i,i'} \prec a = v_B^{i,i'}$, see Fig. 6(b). If $a \in P_t$, we would have $v_B^{i,i'} \prec a \prec v_B^{i,i'}$, which again contradicts the fact that $C_x$ and $C_y$ are equivalent components, see Fig. 6(c).

\[ \square \]

4 Parameterization by Vertex Cover Number

We now turn to the general QUEUE NUMBER problem and show that it is fixed-parameter tractable when parameterized by the vertex cover number. We formalize our result as follows.
Figure 5: Illustration for the proof of Lemma 5: $v_B^{i,i'} v_B^{j,j'}$ nests $ab$. 
Figure 6: Illustration for the proof of Lemma 5: \( v_{i,i'} \) is nested by \( ab \).
Theorem 2 Let $G$ be a graph with $n$ vertices and vertex cover number $\tau = \tau(G)$. A queue layout of $G$ with the minimum number of queues can be computed in $O(2^\tau \cdot \tau \log \tau \cdot n)$ time.

4.1 Algorithm Description

Before describing the algorithm behind Theorem 2, we make an easy observation (which matches an analogous observation in [7]).

Lemma 6 Every $n$-vertex graph $G = (V, E)$ with a vertex cover $C$ of size $\tau$ admits a $\tau$-queue layout. Moreover, if $G$ and $C$ are given as input, such a $\tau$-queue layout can be computed in $O(n + \tau \cdot n)$ time.

Proof: Denote by $c_1, \ldots, c_{\tau}$ the $\tau$ vertices of $C$ and let $\prec$ be any linear order of $G$ such that $c_i \prec c_{i+1}$, for $i = 1, 2, \ldots, \tau - 1$. A queue assignment $\sigma$ of $G$ on $h$ queues can be obtained as follows. Let $U = V \setminus C$. For each $i \in [\tau]$ all edges $u c_i$ with $u \in U \cup \{c_1, \ldots, c_{i-1}\}$ are assigned to queue $i$. Now, consider the edges assigned to any queue $i \in [\tau]$. By construction, they are all incident to vertex $c_i$, and thus no two of them nest each other. Therefore, the pair $(\prec, \sigma)$ is a $\tau$-queue layout of $G$ and can be computed in $O(n + \tau \cdot n)$ time. \qed

Let $C$ be a vertex cover of size $\tau$ of graph $G$. For any subset $U$ of $C$, a vertex $v \in V \setminus C$ is of type $U$ if $N(v) = U$. This defines an equivalence relation on $V \setminus C$ and in particular partitions $V \setminus C$ into at most $\sum_{i=1}^{\tau} \binom{\tau}{i} = 2^\tau - 1 < 2^\tau$ distinct types. Denote by $V_U$ the set of vertices of type $U$.

Lemma 7 Let $h \in \mathbb{N}$ and $v \in V_U$ such that $|V_U| \geq 2 \cdot h^\tau + 2$. Then $G$ admits an $h$-queue layout if and only if $G' = G - \{v\}$ does. Moreover, an $h$-queue layout of $G'$ can be extended to an $h$-queue layout of $G$ in linear time.

The proof of Lemma 7 is deferred to Section 4.2.

Proof: [of Theorem 2] By Proposition 3, we can determine the vertex cover number $\tau$ of $G$ and compute a vertex cover $C$ of size $\tau$ in time $O(2^\tau \cdot \tau \cdot n)$. With Lemma 7 in hand, we can then apply a binary search on the number of queues $h \leq \tau$ as follows. If $h > \tau$, by Lemma 6 we can immediately conclude that $G$ admits a $\tau$-queue layout and compute one in $O(n + \tau \cdot n)$ time. Hence we shall assume that $h \leq \tau$. We construct a kernel $G^*$ from $G$ of size $h^{O(\tau)}$ as follows. We first classify each vertex of $G$ based on its type. We then remove an arbitrary vertex from each set $V_U$ with $|V_U| > 2 \cdot h^\tau + 1$ until $|V_U| \leq 2 \cdot h^\tau + 1$. Thus, constructing $G^*$ can be done in $O(2^\tau + \tau \cdot n)$ time, since $2^\tau$ is the maximum number of types and $\tau \cdot n$ is the maximum number of edges of $G$. From Lemma 7 we conclude that $G$ admits an $h$-queue layout if and only if $G^*$ does.

Given a linear order $\prec^*$ of $G^*$, a queue assignment $\sigma^*$ such that $(\prec^*, \sigma^*)$ is an $h$-queue layout of $G^*$ exists if and only if $\sigma^*$ contains no $h$-rainbow [31], i.e., $h$ independent edges that pairwise nest, which can be easily checked (and computed if it exists) in $h^{O(\tau)}$ time [31]. Consequently, determining whether $G^*$ admits an $h$-queue layout can be done by first guessing all linear orders, and then for each of them by testing for the existence of an $h$-rainbow. Since we have $2^\tau$ types, and each of the at most $2 \cdot h^\tau + 1$ elements of the same type are equivalent in the queue layout (that is, the position of two elements of the same type can be exchanged in $\prec^*$ without affecting $\sigma^*$), the number of linear orders can be upper bounded by $(2^\tau)^{O(h^\tau)} = 2^{\tau^{O(\tau)}}$. Thus, whether $h$ queues suffice for $G^*$ can be determined in $2^{\tau^{O(\tau)}} \cdot h^{O(\tau)} = 2^{\tau^{O(\tau)}}$ time. An $h$-queue layout of $G^*$ (if any) can be extended to one of $G$ by iteratively applying the constructive procedure of Lemma 7,
in $O(\tau \cdot n)$ time. Finally, by applying a binary search on $h$ we obtain an overall time complexity of $O(2^{O(\tau)} + \tau \log \tau \cdot n)$, as desired.

\section*{4.2 Proof of Lemma 7}

One direction follows easily, since removing a vertex from an $h$-queue layout still gives an $h$-queue-layout of the resulting graph. So let $\langle \prec, \sigma \rangle$ be an $h$-queue layout of $G'$. We prove that an $h$-queue layout of $G$ can be constructed by inserting $v$ immediately to the right of a suitable vertex $u$ in $V_U$ and by assigning the edges of $v$ to the same queues as the corresponding edges of $u$.

We say that two vertices $u_1, u_2 \in V_U$ are \textit{queue equivalent}, if for each vertex $w \in U$, the edges $u_1w$ and $u_2w$ are both assigned to the same queue according to $\sigma$. Each vertex in $V_U$ has degree exactly $|U|$, hence this relation partitions the vertices of $V_U$ into at most $h|U| \leq h^2$ sets. Let $V_U^* = V_U \setminus \{v\}$. Since $|V_U^*| \geq 2 \cdot h^2 + 1$, at least three vertices of this set, which we denote by $u_1, u_2$, and $u_3$, are queue equivalent. Consider now the graph induced by the edges of these three vertices that are assigned to a particular queue. By the above argument, such a graph is a $K_{l,3}$, for some $l > 0$. However, $K_{3,3}$ does not admit a 1-page queue layout, because any graph with queue number 1 is planar \cite{32}. As a consequence, $l \leq 2$, that is, each $u_i \in V_U^*$ has at most two edges on each queue. Denote such two edges by $u_iw$ and $u_iz$ and assume without loss of generality $u_1 \prec u_2 \prec u_3$ and $w \prec z$. We now claim that $w \prec u_1 \prec u_2 \prec u_3 \prec z$, else two edges would nest. We can distinguish a few cases based on the position of $u_1$ (recall that $u_1 \prec u_2 \prec u_3$), refer to Fig. 7 for an illustration.

\begin{itemize}
  \item \textbf{Case A:} $w \prec z \prec u_1$, then the nesting edges are $zu_1$ and $wu_2$.
  \item \textbf{Case B:} $u_1 \prec w \prec z$, then we distinguish three more subcases.
    \begin{itemize}
    \item \textbf{Case B.1:} $u_2 \prec w$, then the nesting edges are $u_1z$ and $u_2w$.
    \item \textbf{Case B.2:} $w \prec u_2 \prec z$, then the nesting edges are $u_1z$ and $wu_2$.
    \item \textbf{Case B.3:} $z \prec u_2$, then the nesting edges are $zu_2$ and $wu_3$.
    \end{itemize}
  \item \textbf{Case C:} $w \prec u_1 \prec z$, if $w \prec u_2 \prec u_3 \prec z$ the claim follows. Else, we have two more subcases based again on the position of $u_2$.
\end{itemize}

Figure 7: Illustration for the proof of Lemma 7.
– **Case C.1:** \( w \prec z \prec u_2 \), then the nesting edges are \( wu_2 \) and \( u_1z \).

– **Case C.2:** \( w \prec u_2 \prec z \prec u_3 \), then the nesting edges are \( wu_3 \) and \( u_1z \).

It follows that we can extend \( \prec \) by introducing \( v \) as the first vertex to the right of \( u_1 \) and, for each edge \( vw \) such that \( w \in U \), we can assign \( vw \) to the same queue as \( u_1w \). This operation does not introduce any nesting. Namely, if \( vw \) is assigned to a queue containing only one edge of \( u_1 \), the graph induced by the edges in this queue is a star with center \( w \) and no two edges can nest. If \( vw \) is assigned to a queue containing two edges of \( u_1 \), say \( u_1w \) and \( u_1z \), then we know that all vertices of \( V_U \) are between \( w \) and \( z \) in \( \prec \) and again no two edges nest.

5 Conclusions and Open Problems

We proved that \( h \)-Queue Number is fixed-parameter tractable parameterized by treedepth for \( h = 1 \), and by the vertex cover number for arbitrary \( h \geq 1 \). Several interesting questions arise from our research, among them:

1. A first natural question is to understand whether Theorem 1 can be extended to the general case \( (h \geq 1) \). In particular, our arguments establishing the existence of interleaving components already fail for \( h = 2 \).

2. Extending Theorem 1 to graphs of bounded treewidth is also an interesting problem; here the main issue is to be able to forget information about vertices in a partial order, thus an approach based on testing arched leveled-planarity might be more suitable.

3. Finally, we mention the possibility of studying the parameterized complexity of mixed linear layouts, using both queues and stacks, see \([2,13,25,32,38]\).

It is worth noting that the preliminary version of this manuscript \([8]\) has already led to interesting follow-up work \([34]\) which uses analogous techniques to generalize Theorem 2.

References

[1] J. M. Alam, M. A. Bekos, M. Gronemann, M. Kaufmann, and S. Pupyrev. Lazy queue layouts of posets. In D. Auber and P. Valtr, editors, *GD 2020*, volume 12590 of LNCS, pages 55–68. Springer, 2020. doi:10.1007/978-3-030-68766-3\_5.

[2] P. Angelini, M. A. Bekos, P. Kindermann, and T. Mchedlidze. On mixed linear layouts of series-parallel graphs. In D. Auber and P. Valtr, editors, *GD 2020*, volume 12590 of LNCS, pages 151–159. Springer, 2020. doi:10.1007/978-3-030-68766-3\_12.

[3] M. J. Bannister, S. Cabello, and D. Eppstein. Parameterized complexity of 1-planarity. *J. Graph Algorithms Appl.*, 22(1):23–49, 2018. doi:10.7155/jgaa.00457.

[4] M. J. Bannister, W. E. Devanny, V. Dujmović, D. Eppstein, and D. R. Wood. Track layouts, layered path decompositions, and leveled planarity. *Algorithmica*, 2018. doi:10.1007/s00453-018-0487-5.
[5] M. A. Bekos, H. Förster, M. Gronemann, T. Mchedlidze, F. Montecchiani, C. N. Raftopoulou, and T. Ueckerdt. Planar graphs of bounded degree have bounded queue number. *SIAM J. Comput.*, 48(5):1487–1502, 2019. doi:10.1137/19M125340X.

[6] S. N. Bhatt, F. R. K. Chung, F. T. Leighton, and A. L. Rosenberg. Scheduling tree-dags using FIFO queues: A control-memory trade-off. *J. Parallel Distrib. Comput.*, 33(1):55–68, 1996. doi:10.1006/jpdc.1996.0024.

[7] S. Bhore, R. Ganian, F. Montecchiani, and M. Nöllenburg. Parameterized algorithms for book embedding problems. *J. Graph Algorithms Appl.*, 24(4):603–620, 2020. doi:10.7155/jgaa.00526.

[8] S. Bhore, R. Ganian, F. Montecchiani, and M. Nöllenburg. Parameterized algorithms for queue layouts. In D. Auber and P. Valtr, editors, *GD 2020*, volume 12590 of *LNCS*, pages 40–54. Springer, 2020. doi:10.1007/978-3-030-68766-3_4.

[9] S. Bhore, G. D. Lozzo, F. Montecchiani, and M. Nöllenburg. On the upward book thickness problem: Combinatorial and complexity results. In H. C. Purchase and I. Rutter, editors, *GD 2021*, volume 12868 of *LNCS*, pages 242–256. Springer, 2021. doi:10.1007/978-3-030-92931-2_18.

[10] S. Chaplick, E. Di Giacomo, F. Frati, R. Ganian, C. N. Raftopoulou, and K. Simonov. Parameterized algorithms for upward planarity. In *SoCG 2022*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. To appear.

[11] J. Chen, I. A. Kanj, and G. Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010. doi:10.1016/j.tcs.2010.06.026.

[12] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

[13] P. de Col, F. Klute, and M. Nöllenburg. Mixed linear layouts: Complexity, heuristics, and experiments. In D. Archambault and C. D. Tóth, editors, *GD 2019*, volume 11904 of *LNCS*, pages 460–467. Springer, 2019. doi:10.1007/978-3-030-35802-0_35.

[14] G. Di Battista, F. Frati, and J. Pach. On the queue number of planar graphs. *SIAM J. Comput.*, 42(6):2243–2285, 2013. doi:10.1137/130908051.

[15] E. Di Giacomo, G. Liotta, and H. Meijer. Computing straight-line 3D grid drawings of graphs in linear volume. *Comput. Geom.*, 32(1):26–58, 2005. doi:10.1016/j.comgeo.2004.11.003.

[16] E. Di Giacomo, G. Liotta, and F. Montecchiani. Orthogonal planarity testing of bounded treewidth graphs. *J. Comput. Syst. Sci.*, 125:129–148, 2022. doi:10.1016/j.jcss.2021.11.004.

[17] R. Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[18] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

[19] V. Dujmović. Graph layouts via layered separators. *J. Comb. Theory, Ser. B*, 110:79–89, 2015. doi:10.1016/j.jctb.2014.07.005.
[20] V. Dujmović, M. R. Fellows, M. Kitching, G. Liotta, C. McCartin, N. Nishimura, P. Ragde, F. A. Rosamond, S. Whitesides, and D. R. Wood. On the parameterized complexity of layered graph drawing. *Algorithmica*, 52(2):267–292, 2008. doi:10.1007/s00453-007-9151-1.

[21] V. Dujmović, G. Joret, P. Micek, P. Morin, T. Ueckerdt, and D. R. Wood. Planar graphs have bounded queue-number. In *Foundations of Computer Science (FOCS’19)*, pages 862–875. IEEE, 2019. doi:10.1109/FOCS.2019.00056.

[22] V. Dujmović, P. Morin, and D. R. Wood. Layout of graphs with bounded tree-width. *SIAM J. Comput.*, 34(3):553–579, 2005. doi:10.1137/S0097539702416141.

[23] V. Dujmović, P. Morin, and D. R. Wood. Layered separators in minor-closed graph classes with applications. *J. Comb. Theory, Ser. B*, 127:111–147, 2017. doi:10.1016/j.jctb.2017.05.006.

[24] V. Dujmović and D. R. Wood. On linear layouts of graphs. *Discrete Math. Theor. Comput. Sci.*, 6(2):339–358, 2004. URL: http://dmtcs.episciences.org/317.

[25] V. Dujmović and D. R. Wood. Stacks, queues and tracks: Layouts of graph subdivisions. *Discrete Math. Theor. Comput. Sci.*, 7(1):155–202, 2005. URL: http://dmtcs.episciences.org/346.

[26] E. Eiben, R. Ganian, T. Hamm, F. Klute, and M. Nöllenburg. Extending partial 1-planar drawings. In A. Czumaj, A. Dawar, and E. Merelli, editors, *ICALP 2020*, volume 168 of *LIPIcs*, pages 43:1–43:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.43.

[27] R. Ganian, F. Montecchiani, M. Nöllenburg, and M. Zehavi. Parameterized complexity in graph drawing (dagstuhl seminar 21293). *Dagstuhl Reports*, 11(6):82–123, 2021. doi:10.4230/DagRep.11.6.82.

[28] R. Ganian and S. Ordyniak. The complexity landscape of decompositional parameters for ILP. *Artificial Intelligence*, 257:61–71, 2018. doi:10.1016/j.artint.2017.12.006.

[29] R. Ganian, T. Peitl, F. Slivovsky, and S. Szeider. Fixed-parameter tractability of dependency QBF with structural parameters. In D. Calvanese, E. Erdem, and M. Thielscher, editors, *KR 2020*, pages 392–402, 2020. doi:10.24963/kr.2020/40.

[30] G. Z. Gutin, M. Jones, and M. Wahlström. The mixed Chinese postman problem parameterized by pathwidth and treedepth. *SIAM J. Discrete Math.*, 30(4):2177–2205, 2016. doi:10.1137/15M1034337.

[31] L. S. Heath, F. T. Leighton, and A. L. Rosenberg. Comparing queues and stacks as mechanisms for laying out graphs. *SIAM J. Discrete Math.*, 5(3):398–412, 1992. doi:10.1137/0405031.

[32] L. S. Heath and A. L. Rosenberg. Laying out graphs using queues. *SIAM J. Comput.*, 21(5):927–948, 1992. doi:10.1137/0221055.

[33] P. Hlinený and A. Sankaran. Exact crossing number parameterized by vertex cover. In D. Archambault and C. D. Tóth, editors, *GD 2019*, volume 11904 of *LNCS*, pages 307–319. Springer, 2019. doi:10.1007/978-3-030-35802-0_24.
[34] Y. Liu, Y. Li, and J. Huang. Parameterized algorithms for linear layouts of graphs with respect to the vertex cover number. In D. Du, D. Du, C. Wu, and D. Xu, editors, COCOA 2021, volume 13135 of LNCS, pages 553–567. Springer, 2021. doi:10.1007/978-3-030-92681-6_43.

[35] J. Nešetřil and P. Ossona de Mendez. Sparsity – Graphs, Structures, and Algorithms, volume 28 of Algorithms and combinatorics. Springer, 2012. doi:10.1007/978-3-642-27875-4.

[36] T. Ollmann. On the book thicknesses of various graphs. In Southeastern Conference on Combinatorics, Graph Theory and Computing, volume VIII of Congressus Numerantium, page 459, 1973.

[37] S. V. Pemmaraju. Exploring the powers of stacks and queues via graph layouts. PhD thesis, Virginia Tech, 1992.

[38] S. Pupyrev. Mixed linear layouts of planar graphs. In F. Frati and K.-L. Ma, editors, Graph Drawing and Network Visualization (GD’17), volume 10692 of LNCS, pages 197–209. Springer, 2018. doi:10.1007/978-3-319-73915-1_17.

[39] F. Reidl, P. Rossmanith, F. S. Villaamil, and S. Sikdar. A faster parameterized algorithm for treedepth. In ICALP 2014, volume 8572 of LNCS, pages 931–942. Springer, 2014. doi:10.1007/978-3-662-43948-7_77.

[40] R. E. Tarjan. Sorting using networks of queues and stacks. J. ACM, 19(2):341–346, 1972. doi:10.1145/321694.321704.

[41] V. Wiechert. On the queue-number of graphs with bounded tree-width. Electr. J. Comb., 24(1):P1.65, 2017. doi:10.37236/6429.

[42] M. Yannakakis. Embedding planar graphs in four pages. J. Comput. Syst. Sci., 38(1):36–67, 1989. doi:10.1016/0022-0000(89)90032-9.