A Polynomial-Time Approximation Algorithm for Complete Interval Minors

Romain Bourneuf
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400 Talence, France

Julien Cocquet
ENS de Lyon, Université Claude Bernard Lyon 1, CNRS, Inria, LIP UMR 5668, Lyon, France

Chaoliang Tang
Shanghai Center for Mathematical Science, Fudan University, Shanghai, China
ENS de Lyon, Université Claude Bernard Lyon 1, CNRS, Inria, LIP UMR 5668, Lyon, France

Stéphan Thomassé
ENS de Lyon, Université Claude Bernard Lyon 1, CNRS, Inria, LIP UMR 5668, Lyon, France

Abstract

As shown by Robertson and Seymour, deciding whether the complete graph K_t is a minor of an input graph G is a fixed parameter tractable problem when parameterized by t. From the approximation viewpoint, a substantial gap remains: there is no PTAS for finding the largest complete minor unless $\mathsf{P} = \mathsf{NP}$, whereas the best known result is a polytime $O(\sqrt{n})$-approximation algorithm by Alon, Lingas and Wahlén.

We investigate the complexity of finding K_t as interval minor in ordered graphs (i.e. graphs with a linear order on the vertices, in which intervals are contracted to form minors). Our main result is a polytime $f(t)$-approximation algorithm, where f is triply exponential in t but independent of n. The algorithm is based on delayed decompositions and shows that ordered graphs without a K_t interval minor can be constructed via a bounded number of three operations: closure under substitutions, edge union, and concatenation of a stable set. As a byproduct, graphs avoiding K_t as an interval minor have bounded chromatic number.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph theory

Keywords and phrases Approximation algorithm, Ordered graphs, Interval minors, Delayed decompositions

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2025.15

Category APPROX

Related Version *Full Version*: `https://arxiv.org/abs/2505.05997`

Funding The authors are partially supported by the French National Research Agency under research grants ANR GODASse ANR-24-CE48-4377 and ANR Twin-width ANR-21-CE48-0014-01. The third author is also supported by Chinese Scholarship Council (CSC).

Acknowledgements We thank Julien Duron for his help with the tedious calculations.

1 Introduction

Complete minors in graphs form an extensively studied subject, notably featuring the fundamental result of Robertson and Seymour [27], which asserts that testing whether the complete graph K_t is a minor of an input graph G on n vertices can be done in time $f(t) \cdot n^3$. This was proved in the series of Graph Minors papers, which also provided a decomposition theorem of K_t-minor-free graphs [28] (whose bounds were recently improved significantly by Gorsky, Seweryn and Wiederrecht [16]). One of the key basic facts allowing such a result

is that K_t-minor-free graphs are sparse, i.e. they have a linear number of edges. Recently, Korhonen, Pilipczuk and Stamoulis [22] provided an algorithm running in time $f(t) \cdot n^{1+o(1)}$ for the same problem, improving on a quadratic algorithm from Kawarabayashi, Kobayashi and Reed [19]. From the approximation point of view, the landscape is much less understood. The first hardness result is due to Eppstein [11]: finding the size of a largest complete minor is NP-hard. This was extended by Wahlén [29], who showed that there is no polynomial time approximation scheme (PTAS) for the size of a largest complete minor, unless $\mathsf{P} = \mathsf{NP}$. The current best known approximation factor achievable in polynomial time is $O(\sqrt{n})$, as shown by Alon, Lingas and Wahlén [1]. Up to this day, it is still open whether this problem admits a polytime $f(OPT)$ approximation algorithm. The goal of this paper is to investigate the corresponding problem for complete interval minors in ordered graphs.

An *ordered graph* $(G, <)$ is a graph $G = (V, E)$ equipped with a linear order $<$ on its vertices. We will consistently denote the number of vertices by n, and the number of edges by m. We usually denote the vertices as $v_1, \ldots, v_n$, enumerated according to $<$. For simplicity of notation, we will sometimes omit the order $<$ and talk about an ordered graph G. An *interval minor* of $(G, <)$ is obtained by deleting some edges of G as well as iteratively contracting pairs of vertices which are consecutive in $<$. An example is displayed in Figure 1.

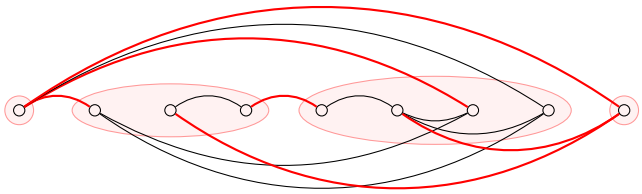

Figure 1 A K_4 interval minor: the red zones represent the intervals we contract and the red edges are the remaining edges. Observe that this interval minor model is not a minor model since we contracted non-connected subsets of vertices.

The central computational problem on interval minors is the following:

Interval Minor Detection
Input: Two ordered graphs G and H.
Question: Is H an interval minor of G?

The complete study of the computational complexity of this question is probably very challenging, and a good first step is to limit ourselves (as for usual minors) to the case where H is a complete graph. The crucial fact to observe is that K_t-interval-minor-free ordered graphs can be very complex. Say that a bipartite ordered graph $(G, <)$ with edges between two parts X and Y is *monotone* if either $X < Y$ or $Y < X$. Then, observe that the monotone $K_{t,t}$ does not contain K_4 as an interval minor. In particular, K_4-interval-minor-free ordered graphs can have a quadratic number of edges. They also form a large class of graphs (with growth at least $2^{n^2/4}$) as they contain all subgraphs of the monotone $K_{n/2,n/2}$. Therefore, any attempt to effectively construct K_t-interval-minor-free ordered graphs must involve an operation allowing the creation of arbitrary monotone bipartite graphs. The landscape of K_t-interval-minor-free ordered graphs thus seems very different from the one of K_t-minor-free graphs. However, a strong connection exists between these two worlds: the right analogy with graph minors involves monotone-$K_{t,t}$-interval-minor-free ordered graphs. To see this, let us first recall the celebrated Marcus-Tardos theorem [24]:

▶ **Theorem 1** ([24]). *There exists a function f such that every ordered graph on n vertices with at least $f(t) \cdot n$ edges contains a monotone $K_{t,t}$ as an interval minor.*

The notion of interval minors was formally introduced by Fox [12] to prove bounds on the function f of Theorem 1. The upper bound was later improved by Cibulka and Kynčl [10], again using interval minors. Theorem 1 gave a positive answer to a conjecture of Füredi and Hajnal [13], stating that for every permutation matrix P, there exists a constant c_P such that every $n \times n$ binary matrix with at least $c_P \cdot n$ entries 1 contains P as a pattern. This result was later generalized to matrices in higher dimension by Klazar and Marcus [21]. The corresponding bound was later improved by Geneson and Tian [15], once more using interval minors. In [12], Fox suggested to conduct a thorough study of ordered graphs avoiding a given interval minor, with the hope of developing a theory analogue to the graph minor theory of Robertson and Seymour. There have been several papers going in that direction, see for instance [25, 23, 20]. In this paper, we continue this line of research by providing a decomposition theorem for K_t-interval-minor-free graphs.

The Marcus-Tardos theorem therefore implies that monotone-$K_{t,t}$-interval-minor-free ordered graphs are sparse, as are K_t-minor-free graphs (in the non-ordered case). A second analogy between these two classes appears from the computational angle:

▶ **Theorem 2** ([17, 5]). *There exists a function g such that testing whether an ordered graph $(G, <)$ contains a monotone $K_{t,t}$ as an interval minor can be done in time $g(t) \cdot |V(G)|$.*

The cornerstone of this algorithm is the FPT algorithm of Guillemot and Marx [17] for detecting a pattern in a permutation. Their algorithm amounts to detecting a monotone $K_{t,t}$ as an interval minor in a monotone matching. The generalization to arbitrary ordered graphs follows from the notion of twin-width introduced in [6]. More specifically, approximating the twin-width of an ordered graph can be done in FPT time, see [5]. In a nutshell, if an ordered graph $(G, <)$ does not contain a monotone complete bipartite graph as an interval minor, then its twin-width is bounded, and therefore dynamic programming can test any first-order formula in FPT time. This gives a win/win algorithm: if the twin-width is large compared to t, simply return Yes since $(G, <)$ must contain $K_{t,t}$ as interval minor, and if not, one can test if $K_{t,t}$ is an interval minor of $(G, <)$, as first-order logic can encode interval minors. Indeed, $(G, <)$ contains an ordered graph $(H, <')$ on vertex set $u_1 <' \ldots <' u_h$ if and only if there exist vertices $x_1, \ldots, x_{h-1} \in V(G)$ such that $x_1 < \ldots < x_{h-1}$, and for every edge $u_i u_j \in E(H)$, there exists $y_i, y_j \in V(G)$ such that $x_{i-1} < y_i \leq x_i$, $x_{j-1} < y_j \leq x_j$ and $y_i y_j \in E(G)$ (with the obvious adaptation for u_1 and u_h). Let us end this detour about $K_{t,t}$ interval minors with the central open question in this topic: Given a graph G which admits an order $<$ avoiding $K_{t,t}$ as an interval minor, can we efficiently compute an order $<'$ such that $(G, <')$ avoids $K_{f(t),f(t)}$ as an interval minor. This question is equivalent to ask for a $f(OPT)$-approximation algorithm of the twin-width for sparse graphs (see Theorem 2.12 in [4]).

Let us go back to our original goal of approximately detecting K_t as an interval minor. We follow the classical strategy of providing an effective decomposition of K_t-interval-minor-free ordered graphs using a bounded number of tractable operations. The key-tool for this is the notion of *delayed decomposition*, used in [26] and formally defined in [7] to show that graphs of bounded twin-width are polynomially χ-bounded. It was later used as the central tool to show that pattern-avoiding permutations are the product of a bounded number of separable permutations [3]. We first introduce three operations on classes of ordered graphs. Given two classes $\mathcal{C}, \mathcal{C}'$ of ordered graphs, we define:

- The *substitution closure* of $\mathcal{C}$ is the class $\overline{\mathcal{C}}$ of ordered graphs which can be obtained from $\mathcal{C}$ by iterating an arbitrary number of substitutions of elements of $\mathcal{C}$. The *substitution* of a vertex v of an ordered graph $(G, <)$ by an ordered graph $(H, <')$ consists of replacing v by a copy of $(H, <')$ and joining all neighbors of v to all vertices of H. The vertices of H remain ordered according to $<'$, at the position of v in $<$.
- The *edge union* of $\mathcal{C}$ and $\mathcal{C}'$ is the class $\mathcal{C} \oplus \mathcal{C}'$ of all ordered graphs of the form $(G \oplus G', <)$ where $(G, <) \in \mathcal{C}$ and $(G', <) \in \mathcal{C}'$ are two graphs on the same set of vertices ordered by the same linear order $<$, and $G \oplus G'$ is their edge union.
- The *independent concatenation* of $\mathcal{C}$ is the class $\mathcal{C}^+$ of all ordered graphs $(G^+, <^+)$ which can be obtained from a graph $(G, <) \in \mathcal{C}$ by adding an independent set I at the beginning or at the end of $<$, which can be connected arbitrarily to the vertices of G. Formally, either $V(G) <^+ I$ or $I <^+ V(G)$.

We define the *rank* of an ordered graph $(G, <)$ inductively, as follows:

- The empty graph has rank 0.
- The rank of $(G, <)$ is the smallest integer r such that $(G, <)$ can be built from graphs of rank at most $r - 1$, using one of the following operations: substitution closure, edge union or independent concatenation.

Let us illustrate this parameter. The class of ordered edgeless graphs has rank 1, as they can be obtained by an independent concatenation from the empty graph. The class of monotone bipartite graphs has rank 2, as we can perform another round of independent concatenation. In particular, this means that the class of ordered graphs with rank at most 2 has unbounded twin-width, and even unbounded VC-dimension. The class of ordered complete graphs has rank at most 3, since the edge graph K_2 is obtained at rank 2, and its closure under substitution gives all complete graphs. A slightly more involved example is that of the ordered path $(P_n, <)$, in which each vertex is joined to its successor in $<$.

Note that the subgraph M_o of P_n consisting of the odd indexed edges is a matching consisting of edges $e_1 < e_3 < \dots$, hence can obtained by substituting the vertices of an independent set by edges. In particular M_o has rank 3, and the even indexed edges subgraph M_e also has rank 3. Finally, $P_n = M_o \oplus M_e$ has rank 4. This is illustrated in Figure 2.

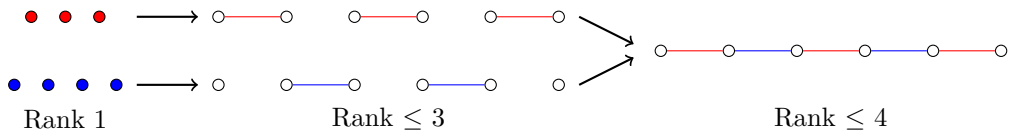

Figure 2 Construction showing that the ordered P_6 has rank at most 4. The edge graph K_2 has rank 2, which is why we go from rank 1 to rank ≤ 3.

The central result of this paper is the following:

▶ **Theorem 3.** *Every K_t-interval-minor-free ordered graph $(G, <)$ has bounded rank.*

The cornerstone of Theorem 3 is the interplay between the two operations of substitution closure and edge union (as is the case for the construction of a path from two matchings). Before introducing the notion of delayed decomposition, let us first formalize what it means exactly that a graph G belongs to the substitution closure of a class $\mathcal{C}$. The adequate representation of G must take into account that some vertices have been repeatedly substituted by a graph of $\mathcal{C}$, hence G can be expressed as a tree of substitutions. Formally, a *$\mathcal{C}$-substitution*

tree of G is a rooted tree T whose leaves are the vertices of G. Moreover, every internal node x of T is labeled by a graph G_x of $\mathcal{C}$, whose vertices are the children of x in T. Finally, two vertices u, v of G form an edge if and only if given their closest common ancestor x, the two children u', v' of x which are the respective ancestors of u, v form an edge in G_x. By construction, the graphs in the substitution closure of $\mathcal{C}$ are precisely the ones representable by a $\mathcal{C}$-substitution tree. These structured trees were introduced by Gallai to study partial orders [14]. The most popular examples of substitution trees are those used to decompose cographs (P_4-free graphs) using binary trees in which the graphs G_x are the edge and the non edge. These trees are well-suited for ordered graphs, as the order $<$ can be represented as the left-to-right order on the leaves.

However, most graphs G do not admit a non-trivial substitution tree, i.e. one in which the root is not labeled by G. For instance paths of length at least 3 are indecomposable (or prime) with respect to substitutions. A simple way to strengthen substitution trees is to delay the effect of the graphs G_x: instead of creating edges between children, they act on their grandchildren. Let us formalize this:

A *delayed structured tree* is a rooted tree T whose leaves are the vertices of a graph G (the *realization* of T). Moreover, every internal node x of T is labeled by a *quotient graph* G_x whose vertices are the grandchildren of x in T. Finally, two vertices u, v of G are connected if and only if given their closest common ancestor x, the two grandchildren u', v' of x which are the respective ancestors of u, v are connected in G_x. For an example, see Figure 4. Authorizing this delay results in a tool which is much more expressive than substitution trees. In particular, given a class of graphs $\mathcal{C}$, the class $\mathcal{C}'$ of realizations of delayed structured trees whose quotient graphs G_x belong to $\mathcal{C}$ is much harder to grasp than the mere substitution closure of $\mathcal{C}$.

However, the class $\mathcal{C}'$ is not too complex compared to $\mathcal{C}$: given the delayed structured tree T, consider the quotient graphs G_x of the nodes x with even depth, and of the ones with odd depth. Now form two trees T_e and T_o from T, by setting in T_e all quotient graphs G_x at odd depth as edgeless graphs, and setting in T_o all quotient graphs G_x at even depth as edgeless graphs. The key observation is that the realization G_e of T_e belongs to the substitution closure of $\mathcal{C}$. The same holds for G_o, and thus G is the edge union of two graphs in the substitution closure of $\mathcal{C}$.

In a nutshell, the proof of Theorem 3 is now straightforward: we just have to show that every ordered graph $(G, <)$ without K_t-interval minor is the realization of a delayed structured tree whose quotient graphs are *simpler* than G, and that these simpler graphs are again decomposable into simpler objects, and that this process has a bounded height of recursion. This is the main technical part of the paper. Note that this process is too tame to create high-entropy objects such as monotone bipartite graphs. Here a choice has to be made: either declare that the basic class is indeed all monotone bipartite graphs, and just consider substitution closure and edge union, or set the empty graph as our basic class, and authorize independent set concatenation. We chose the latter convention as it fits more in our algorithmic purpose, but we feel that the former is more in the spirit of classical graph decompositions, with only two tame operations, and the basic class consisting of the indecomposable (or prime) monotone bipartite graphs.

The proof of Theorem 3 is algorithmic, and a sequence of operations for constructing $(G, <)$ can be effectively computed in polynomial time by iterating delayed decompositions. This is the first phase of our algorithm to detect K_t as an interval minor: either we fail to achieve bounded rank and then find K_t as an interval minor, or we compute a sequence of operations achieving bounded rank for $(G, <)$. Unfortunately, a graph can have bounded

rank and still contain a K_t interval minor. Indeed, as we saw before, complete graphs have rank at most 3. The nice point is that large complete subgraphs are the only reason why an ordered graph of bounded rank can contain arbitrarily large complete interval minors. The second phase of the algorithm uses the decomposition of $(G, <)$ and either outputs a K_t subgraph, or certifies that G has no $K_{f(t)}$ interval minor. As a result, we show:

▶ **Theorem 4.** *There is a triply exponential function f and a decision algorithm which, given as input an ordered graph $(G, <)$ with n vertices and m edges and an integer t, satisfies the following:*

- *If the algorithm returns Yes then $(G, <)$ contains K_t as an interval minor.*
- *If the algorithm returns No then $(G, <)$ does not contain $K_{f(t)}$ as an interval minor.*
- *The algorithm runs in time $O(t \cdot mn^2)$.*

Furthermore, when the algorithm returns Yes, it can also output a model of a K_t interval minor within the same running time.

The bound on the approximation factor is primarily due to the second phase of the algorithm. Specifically, we need to argue that if G has a huge complete interval minor and small rank, then we can detect a K_t subgraph in G.

First, observe that if G can be obtained by independent concatenation from G' then G' still contains a huge complete interval minor. Similarly, if G is the edge union of G_1 and G_2 then it follows from Ramsey's theorem that one of G_1 and G_2 still contains a large complete interval minor. However, since we will apply this argument repeatedly, the standard version of Ramsey's theorem would yield a tower-type bound. To avoid this, we establish a Ramsey-type result for interval minors (see Section 4.2), which eventually allows us to reduce the bound to exponential-type. Finally, if G is obtained by substitution closure then either G contains a clique which can be easily detected from the delayed decomposition tree, or one of the quotient graphs still contains a large complete interval minor (see Section 2.3).

Before diving into the details, we now discuss two hardness results in the context of K_t-minors.

Hardness of ordering a graph

A natural question regarding interval minors in ordered graphs is to find the minimum t, such that an input graph G admits an ordering $<$ with no K_t interval minor. Denote this value by $kim(G)$. This question is particularly interesting since the same problem for $K_{t,t}$ instead of K_t amounts to approximating the twin-width of a sparse graph. Unfortunately the answer for K_t is as hard as approximating the chromatic number χ of a graph:

▶ **Lemma 5.** *The parameters $kim(G)$ and $\chi(G)$ are functionally equivalent.*

Proof. If a graph has chromatic number t, ordering its vertices according to the color classes directly gives an order without K_{2t} interval minor. To see it, consider any partition of the vertices into $2t$ intervals. At most $t-1$ intervals can contain vertices from two different color classes, so at least $t+1$ intervals must be entirely contained in a color class. By the pigeonhole principle, two distinct intervals must be contained inside the same color class, and therefore cannot be connected by an edge. Conversely, if a graph G has an ordering $<$ such that $(G, <)$ does not contain a K_t interval minor, Theorem 3 implies that $(G, <)$ has bounded rank. Say that a class $\mathcal{C}$ of graphs is *χ-bounded* if there exists a function h such that every graph G in $\mathcal{C}$ satisfies $\chi(G) \leq h(\omega(G))$, where $\omega(G)$ is the maximum size of a clique of G. We speak of *polynomial* χ-boundedness if h is a polynomial.

▷ Claim. For every $r \geq 0$, the class $\mathcal{C}_r$ of ordered graphs with rank at most r is polynomially χ-bounded.

Proof of the Claim. We prove it by induction on r. The statement is trivial for $r = 0$ since the empty graph has clique number 0 and chromatic number 0. For the induction step, we just have to show that the three operations preserve polynomial χ-boundedness. Independent concatenation increases the chromatic number by at most one and cannot decrease the clique number. Similarly, the chromatic number of the edge union of two graphs is at most the product of their chromatic numbers, and the clique number of the edge union of two graphs is at least the maximum of the two clique numbers. The fact that the substitution closure preserves polynomial χ-boundedness was shown by Chudnovsky, Penev, Scott and Trotignon [9]. ◁

Then, if $(G, <)$ does not contain a K_t interval minor then G does not contain a clique of size t, and therefore its chromatic number is bounded. ◀

The fact that K_t-interval-minor-free ordered graphs have bounded chromatic number directly implies that the class of ordered graphs which do not contain some (arbitrary but fixed) ordered matching as a subgraph has bounded chromatic number (the two statements are in fact easily equivalent). This problem was introduced in [2] by Axenovich, Rollin and Ueckerdt, where they ask the question for some specific matchings. A far-reaching generalization was announced by Briański, Davies and Walczak [8]: for any ordered matching M, the class of ordered graphs which do not contain M as an induced ordered subgraph is χ-bounded. A study of the list chromatic number of ordered graphs avoiding an induced subgraph was also conducted by Hajebi, Li and Spirkl [18].

Hardness of detecting complete interval minors in ordered graphs

In our main result, we give a polynomial approximate algorithm to detect whether an ordered graph contains a K_t interval minor. For the hardness part, we show that deciding whether the complete interval minor number is at least t is NP-complete in general. Formally, we consider the following problem.

Complete Interval Minor
Input: An ordered graph G and an integer t.
Question: Is K_t an interval minor of G?

▶ **Theorem 6.** *Complete Interval Minor is* NP*-complete.*

Proof. We first show that the problem is in NP. To do so, observe that we can describe a K_t interval minor model of G by giving the intervals. Then, it can easily be checked in polynomial time that these intervals indeed form a model of G.

We show that the problem is NP-hard by reduction from Clique. Consider an instance (G, k) of Clique: we are asked whether G contains a clique of size at least k. We build an instance $(\hat{G}, t)$ of Complete Interval Minor as follows. Write $V(G) = \{v_1, \ldots, v_n\}$. Consider the ordered graph $(\hat{G}, <)$ defined as

- $V(\hat{G}) = \{v_1, \ldots, v_n\} \cup \{u_1, \ldots, u_{n-1}\}$, where the u_i are fresh vertices,
- $v_i v_j \in E(\hat{G})$ if and only if $v_i v_j \in E(G)$, $v_i u_j \in E(\hat{G})$ for every $i \in [n]$ and $j \in [n-1]$, and $u_i u_j \in E(\hat{G})$ for every $i \neq j \in [n-1]$,
- $v_1 < u_1 < v_2 < u_2 < \ldots < v_{n-1} < u_{n-1} < v_n$.

Informally, the ordered graph $(\hat{G}, <)$ is obtained from G by arbitrarily ordering the vertices of G, and then adding a universal vertex between any two vertices of G. See Figure 3 for an illutration. Finally, we set $t = n - 1 + k$. Note that $((\hat{G}, <), t)$ can be built from (G, k) in polynomial time.

We now prove that G has a clique of size k if and only if K_t is an interval minor of $(\hat{G}, <)$. Suppose first that G has a clique of size k induced by the vertices $v_{i_1}, \ldots, v_{i_k}$. Then, the vertices $v_{i_1}, \ldots, v_{i_k}, u_1, \ldots, u_{n-1}$ induce a clique of size $k + n - 1 = t$ in $\hat{G}$, hence $(\hat{G}, <)$ contains K_t as an interval minor. Conversely, suppose that K_t is an interval minor of $(\hat{G}, <)$. Since there are only $n - 1$ vertices u_j, there are at least $t - (n - 1) = k$ intervals in the interval model of K_t which do not contain a vertex u_j. By definition of the order $<$, any interval that does not contain a vertex u_j is of the form $\{v_i\}$. Let $\{v_{i_1}\}, \ldots, \{v_{i_k}\}$ be these k intervals. Then, the vertices $v_{i_1}, \ldots, v_{i_k}$ induce a clique of size k in G. ◀

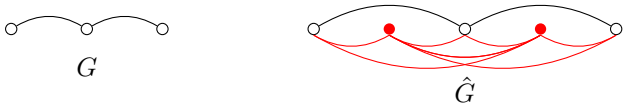

Figure 3 A graph G drawn with an arbitrary order and the corresponding ordered graph $\hat{G}$. The fresh vertices u_1 and u_2 are depicted in red, as well as the edges incident to them.

Perspectives

The obvious next step is of course to obtain a more digestible approximation factor than the triply exponential function f. We did not try very hard to show lower bounds, but failed to rule out constant factor approximation. A very natural problem to ask is the existence of an FPT algorithm to find a K_t interval minor. This looks really challenging, as the only strategy seems to obtain a much more precise description of K_t-interval-minor-free ordered graphs which would allow dynamic programming. One of the main conclusions of this study is the versatility of delayed decompositions. It really suffices to apply them in a canonical way, and wonder whether the quotient graphs are simpler than the original one. For which (not necessarily ordered) graph classes does this machinery provide a bounded rank decomposition?

Organization of the paper

In Section 2, we formally define delayed decompositions, show that they can be computed efficiently, and review some of their properties. Then, in Section 3, we introduce a variant of rank tailored for algorithmic use, the *delayed rank*, which is based on delayed decompositions. We study some of its basic properties, before proving that ordered graphs with large delayed rank contain large complete interval minors. Finally, in Section 4, we present the algorithm of Theorem 4. To bound its approximation factor, we prove a Ramsey-type result for interval minors.

2 Delayed decomposition

This section focuses on the notion of delayed decomposition, a key tool for our approximation algorithm. We start by introducing delayed decompositions in Section 2.1. Then, in Section 2.2, we prove that all (ordered) graphs admit so-called distinguishing delayed

decompositions, and that they can be computed in linear time. Finally, in Section 2.3, we establish a variant of Kőnig's lemma on trees, and explore its consequences related to delayed decompositions.

2.1 Definition

We consider rooted trees T, and call the vertices of T *nodes*. The *ancestors* of a node x are the nodes in the unique path from x to the root r of T, and the *parent* of x is the first node on this path (other than x). The root has no parent. We also speak of *grandparents*, *descendants*, *children*, *grandchildren*, *siblings* (nodes with same parent), and *cousins* (non-sibling nodes with the same grandparent). The set of leaves of a tree T is denoted by $L(T)$. For any node $x \in V(T)$, we denote by $L(x) \subseteq L(T)$ the set of leaves which are descendants of x.

An *ordered tree* $(T, <)$ is a rooted tree T equipped with a linear order $<$ on $L(T)$, such that for each node $x \in V(T)$, the leaves $L(x)$ form an interval of $<$. It is natural to think of $<$ as a *left-to-right* order on the leaves of T. If $(T, <)$ is an ordered tree and $x, y \in V(T)$ are not in an ancestor–descendant relationship, then $L(x)$ and $L(y)$ are disjoint, and each of them is an interval for $<$, hence either $L(x) < L(y)$, or $L(y) < L(x)$. We then naturally extend $<$ to x, y by $x < y$ in the former case, and $y < x$ in the latter. In particular, $<$ induces a linear order $<_x$ on the children of any internal node x. Therefore, given a child y of a node x, we can speak of the *predecessor* and the *successor* of y, and of *consecutive* children of x. We can also consider the *first child* of x, which is the $<_x$-minimum child of x, and the *last child* of x, defined analogously.

A *delayed structured tree* $\left(T, <, \{G_x\}_{x \in V(T)}\right)$ is an ordered tree $(T, <)$, equipped with, for each node $x \in V(T)$, a graph G_x on the *grandchildren* of x. We will often refer to these graphs G_x as the *quotient graphs*. This is analogous to the trees describing substitutions (module decomposition trees), except that the graphs G_x are defined on the grandchildren instead of the children, hence "delayed". We add the technical requirement that each leaf is a single child (with no siblings), so that whenever $x \neq y$ are leaves, their closest ancestor is at distance at least 2.

The *realization* of a delayed structured tree $\left(T, <, \{G_x\}_{x \in V(T)}\right)$ is the ordered graph $G_T = ((L(T), E_T), <)$, where for two leaves x, y of T, we have the edge $xy \in E_T$ if and only if $x'y' \in E(G_z)$, where z is the closest ancestor of x, y, and x', y' are the grandchildren of z which are ancestors of x, y respectively. If $(G, <)$ is the realization $\left(T, <, \{G_x\}_{x \in V(T)}\right)$, we say that $\left(T, <, \{G_x\}_{x \in V(T)}\right)$ is a *delayed decomposition* of $(G, <)$. See Figure 4 for an example.

A delayed decomposition $\left(T, <, \{G_x\}_{x \in V(T)}\right)$ of an ordered graph $(G, <)$ is *distinguishing* if it satisfies the following property: For every node x which has at least 3 children, for every two consecutive children x_1, x_2 of x, there exists some vertex $v \in V(G) \setminus L(x)$ which is adjacent to all vertices in one of $L(x_1), L(x_2)$ and to no vertex in the other.

As we shall see in Theorem 8, every graph has a distinguishing delayed decomposition, which can be computed in linear time. When we talk about *the* distinguishing delayed decomposition of a graph, we mean the one computed by Theorem 8.

The following result was already observed in [7].

▶ **Lemma 7** ([7, Lemma 2.1]). *Let $(G, <)$ be the realization of a delayed structured tree $\left(T, <, \{G_x\}_{x \in V(T)}\right)$. Then, $(G, <)$ is the edge union of two ordered graphs, each of which can be obtained by substitutions from the quotient graphs $\{G_x\}_{x \in V(T)}$.*

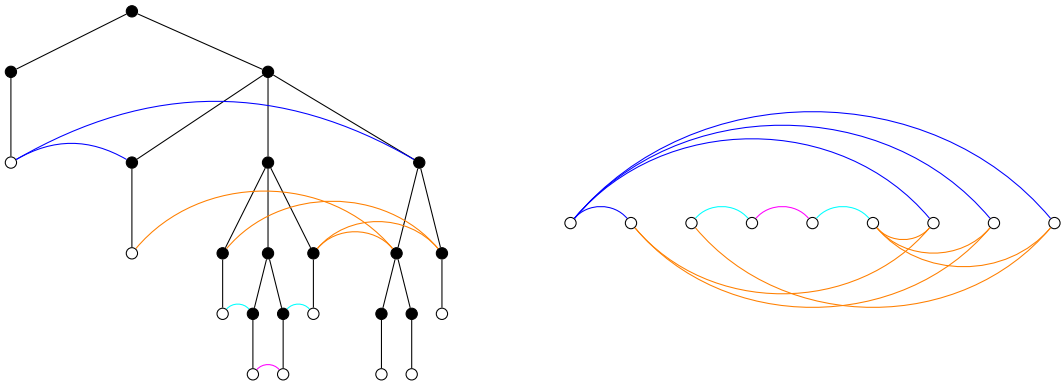

Figure 4 A delayed structured tree and its realization. The edges in the realization are colored according to their corresponding quotient graph.

2.2 Computing a distinguishing delayed decomposition

In this section, we prove that every ordered graph has a distinguishing delayed decomposition, which can be computed in linear time.

To discuss the running time of the algorithm, we first detail how we store ordered graphs. Throughout this article, we assume that the edge set is stored as a list of edges, and the vertex set as a list of vertices. There are two natural ways to store the order on the vertex set. The first one is to store the *ordered* set of vertices explicitly, for instance by assuming that the list of vertices is sorted according to the order. In particular, after a $O(n)$-time precomputation, all vertices can store their rank in the order. We call this representation *explicit*. The second one is to assume that the order can be determined from the labels of the vertices, which is the case for instance if the vertices are ordered by increasing labels. We call this representation *implicit*. Observe that an explicit representation can be computed in time $O(n \log(n))$ from an implicit representation by sorting the vertex set according to the order, and then storing the sorted list explicitly. Given an explicit representation of an ordered graph, we can relabel all the vertices in time $O(n)$ so that the vertex set is $[n]$, equipped with its natural linear order.

An algorithm to compute a distinguishing delayed decomposition of an ordered graph was described in [7], where delayed decompositions were formally introduced, and it was shown in [3] how to implement this algorithm in linear time in the context of permutations. A straightforward adaptation to the context of ordered graphs yields the following result.

▶ **Theorem 8** ([3]). *There is an algorithm which, given as input an explicit ordered graph $(G, <)$ with n vertices and m edges, computes in time $O(m + n)$ a distinguishing delayed decomposition $\left(T, <, \{G_x\}_{x \in V(T)}\right)$ of $(G, <)$.*

▶ Remark 9. If $(G, <)$ is an ordered graph with n vertices and m edges, the distinguishing delayed decomposition computed by Theorem 8 has the property that the total number of edges over all quotient graphs is at most m. Furthermore, the number of quotient graphs is equal to the number of nodes of T which have grandchildren, which is at most $n - 1$. By construction, for every node x of T and every child y of x, the children of y form an independent set in G_x.

2.3 A result on trees and its consequences

A classical result on trees states that every tree with a very large number of leaves contains either a node with large degree, or a path containing many nodes of degree at least 3. The goal of this section is to prove a generalization of this statement in the context of ordered trees. More precisely, consider an ordered tree whose leaves are grouped into disjoint intervals. We prove that if there is a very large number of intervals, either there is a node which "splits" a large number of intervals between its children, or there is a long path which progressively "peels" a large number of intervals.

Given an ordered tree $(T, <)$, an *interval family* of $(T, <)$ is a set $\mathcal{I}$ of disjoint intervals of the linear order $(L(T), <)$. A node $x \in V(T)$ is *b-branching* if there is a subset $\mathcal{I}' \subseteq \mathcal{I}$ of b intervals such that every interval of $\mathcal{I}'$ is included in $L(x)$ and there is no child y of x such that $L(y)$ intersects two intervals of $\mathcal{I}'$. See Figure 5 for an illustration. An *ℓ-interval path* is a sequence of intervals $I_1, \dots, I_\ell$ such that there exists nodes $x_1, \dots, x_\ell$ of T such that:

- $L(x_j)$ contains $I_j \cup \dots \cup I_\ell$ for all $j = 1, \dots, \ell$.
- $L(x_j) \cap I_{j-1} = \emptyset$ for all $j = 2, \dots, \ell$.

Note that the nodes x_j are pairwise distinct, and x_j is a descendant of x_i whenever $i < j$. Also, when $j \geq 3$, the parent $p(x_j)$ of x_j satisfies $L(p(x_j)) \cap I_{j-2} = \emptyset$ since $p(x_j)$ is a (not necessarily proper) descendant of x_{j-1}.

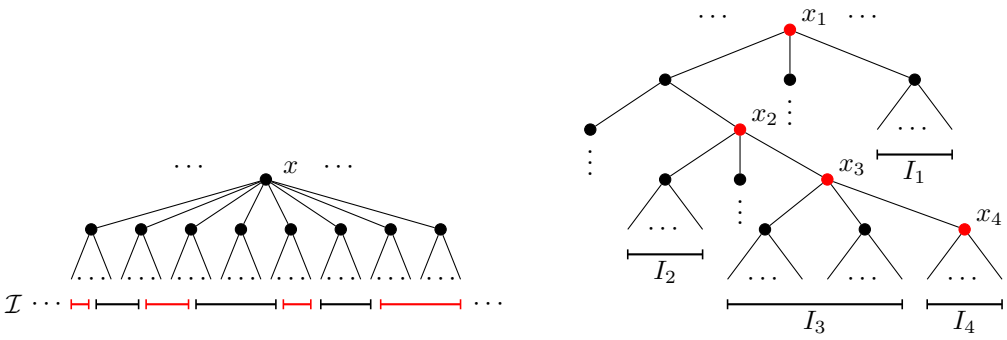

Figure 5 A 4-branching vertex x, with $\mathcal{I}'$ being the set of red intervals, and a 4-interval path I_1, I_2, I_3, I_4.

▶ **Lemma 10.** *Given an interval family $\mathcal{I}$ of size $2(b+2)^t$ of an ordered tree $(T, <)$, there is either a b-branching node or a t-interval path in T.*

Proof. For every node x of T, let $w(x)$ be the number of intervals of $\mathcal{I}$ which are entirely contained inside $L(x)$. Let $B(x)$ be the set of children y of x such that $w(y) \neq 0$. Note that x is (at least) $|B(x)|$-branching, and so we can conclude if $|B(x)| \geq b$. Therefore, we can assume that $|B(x)| < b$ for every $x \in V(T)$.

Consider a node x with $w(x) \geq 2b^2$. Let $\mathcal{I}_1$ be the set of intervals which are contained inside some $L(y)$ for $y \in B(x)$ and $\mathcal{I}_2$ be the set of intervals which are contained inside $L(x)$ but are not in $\mathcal{I}_1$. Note that if $|\mathcal{I}_2| \geq 2b - 1$, the node x is b-branching. Indeed, in that case it suffices to order $\mathcal{I}_2$ with respect to $<$, and to select every other interval to form a b-branching subfamily $\mathcal{I}'$.

So there is a child y of x such that (using that $w(x) \geq 2b^2$ in the last inequality):

$$w(y) \geq |\mathcal{I}_1|/|B(x)| \geq (w(x) - 2b + 1)/(b-1) \geq w(x)/b.$$

Starting from the root r, we define a sequence of vertices $(r = x_1, \dots, x_t)$ forming a descending chain in T, such that $w(x_{i+1}) \leq w(x_i) - 3$ for every $i \in [t-1]$ and $w(x_i) \geq 2(b+2)^{t+1-i}$ for every $i \in [t]$.

Start by setting x_1 to be the root r, and note that $w(x_1) = 2(b+2)^t$. Suppose that we already defined $x_1, \dots, x_{i-1}$ satisfying the desired property, with $i \leq t$. Consider a descendant x_i of x_{i-1} with maximum $w(x_i)$, such that $w(x_i) \leq w(x_{i-1}) - 3$, and among them one which is as close to the root as possible in T. By definition of x_i, we have:

$$w(p(x_i)) \geq w(x_{i-1}) - 2 \geq 2(b+2)^{t+2-i} - 2 \geq 2(b+2)^2 - 2 \geq 2b^2.$$

Thus, $p(x_i)$ has a child y with

$$w(y) \geq w(p(x_i))/b \geq (w(x_{i-1}) - 2)/b.$$

Therefore, by maximality of $w(x_i)$, we have:

$$w(x_i) \geq (w(x_{i-1}) - 2)/b \geq (2(b+2)^{t+2-i} - 2)/b \geq 2(b+2)^{t+1-i}.$$

For every $i \in [t-1]$, since $w(x_{i+1}) \leq w(x_i) - 3$, there are at least three intervals which are entirely contained in $L(x_i)$ and which are not entirely contained in $L(x_{i+1})$. Therefore, there is an interval I_i which is entirely contained in $L(x_i)$ and such that $I_i \cap L(x_{i+1}) = \emptyset$. Finally, $w(x_t) \geq 2(b+2) > 0$ so there exists an interval $I_t \subseteq L(x_t)$. ◀

If $(T, <, \{G_x\}_{x \in V(T)})$ is a delayed structured tree, a leaf y of T is *h-heavy* if there are at least h ancestors x of y which are not isolated in the graph $G_{p^2(x)}$, where $p^2(x)$ is the grandparent of x in T.

▶ **Lemma 11.** *Let $(T, <, \{G_x\}_{x \in V(T)})$ be a delayed structured tree and $(G_T, <)$ be its realization. Let $\mathcal{I}$ be an interval family of $(T, <)$ forming a complete interval minor in $(G_T, <)$. If $\mathcal{I}$ has a $(2t-1)$-interval path in T, then there is a $(2t-3)$-heavy leaf in $(T, <, \{G_x\}_{x \in V(T)})$.*

Proof. Let $I_1, \dots, I_{2t-1}$ be the intervals of the $(2t-1)$-interval path of $\mathcal{I}$ in T, and $x_1, \dots, x_{2t-1}$ be nodes of T such that for all $j \in [2t-1]$, $L(x_j)$ contains $I_j \cup \dots \cup I_{2t-1}$, and for all $j \in [\![2, 2t-1]\!]$, $L(x_j) \cap I_{j-1} = \emptyset$.

Let $y \in L(T)$ be any leaf in I_{2t-1}. We show that y is $(2t-3)$-heavy. Let $i \in [2t-3]$. Since $\mathcal{I}$ forms a complete interval minor in $(G_T, <)$, there is an edge between some vertex $y_i \in I_i$ and I_{2t-1}. Let z_i be the deepest node of T such that $I_{2t-1} \cup \{y_i\} \subseteq L(z_i)$. Then, z_i is a descendant of x_i and a proper ancestor of x_{i+1} (hence all z_i are distinct). Furthermore, since z_i is a proper ancestor of x_{i+1} then the grandchild w_i of z_i such that $y \in L(w_i)$ is an ancestor of x_{i+2}. In particular, $I_{2t-1} \subseteq L(x_{2t-1}) \subseteq L(x_{i+2}) \subseteq L(w_i)$. Let w_i' be the grandchild of z_i such that $y_i \in L(w_i')$. By maximality of the depth of z_i, w_i and w_i' are cousins in T. Since there is an edge between y_i and I_{2t-1} in G_T, there is an edge between w_i and w_i' in G_{z_i}. Thus, we found $2t-3$ distinct ancestors w of y which are not isolated in $G_{p^2(w)}$, i.e. y is $(2t-3)$-heavy. ◀

▶ **Lemma 12.** *Let $(T, <, \{G_x\}_{x \in V(T)})$ be a delayed structured tree containing a $(2t-3)$-heavy leaf. Then, its realization $(G_T, <)$ contains a clique of size t as a subgraph.*

Proof. Let y be a $(2t-3)$-heavy leaf of T. Let $x_1, \dots, x_{2t-3}$ be ancestors of y such that each x_i is not isolated in the graph $G_{p^2(x_i)}$. Up to renaming them, we can assume that x_i is a proper ancestor of x_j whenever $i < j$. We build a sequence $(y_0, \dots, y_{t-1})$ of vertices of G_T with the following properties:

- For every $i \in [\![0, t-2]\!]$, y_i is adjacent to every vertex in $L(x_{2i+1})$, and
- For every $i \in [t-1]$, $y_i \in L(x_{2i-1})$.

Note that these properties immediately imply that $y_0, \dots, y_{t-1}$ induce a clique of size t in G_T.

Let $i \in [\![0, t-2]\!]$ and let x'_i be a neighbor of x_{2i+1} in the graph $G_{p^2(x_{2i+1})}$. Let y_i be an arbitrary vertex in $L(x'_i)$. Then, y_i is adjacent to all vertices in $L(x_{2i+1})$. If $i > 0$ then $p^2(x_{2i+1})$ is a descendant (not necessarily proper) of x_{2i-1} so $y_i \in L(x_{2i-1})$. Finally, we choose y_{t-1} to be any vertex of $L(x_{2t-3})$. ◀

3 Delayed rank

The notion of rank is natural and convenient for structural analysis, but not so much for algorithmic purposes, since there does not seem to be a simple way of computing it, or even approximating it. To overcome this issue, we introduce an alternative to the rank, based on delayed decompositions, which we call the *delayed rank*. This delayed rank can easily be computed, and shares some key structural properties with the rank. It is the key notion for our approximation algorithm.

3.1 Definition and first properties

We add some information to the delayed structured trees. Consider a delayed structured tree $(T, <, \{G_x\}_{x \in V(T)})$. We label the nodes x of T as follows (see Figure 6):

- If x does not have a grand-parent, we label it with $\emptyset$ (the first two layers of the tree).
- Else, if there is a cousin x' of x such that $x < x'$ and there is an edge xx' in $G_{p^2(x)}$ then we label x with R.
- Otherwise, if there is a cousin x' of x such that $x' < x$ and there is an edge xx' in $G_{p^2(x)}$ then we label x with L.
- Else, we label x with O.

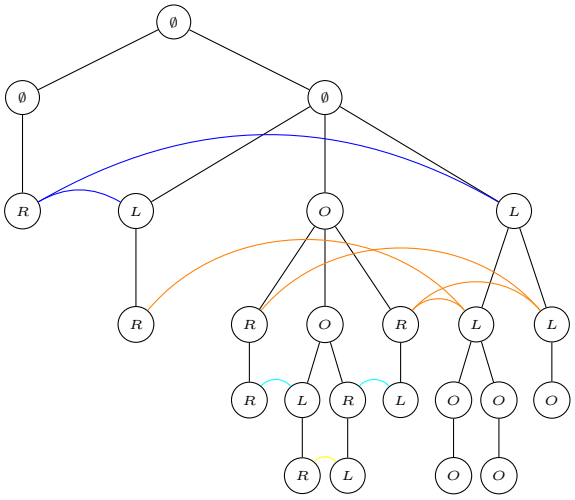

Figure 6 Labeling of a delayed structured tree.

▶ **Lemma 13.** *Let $(T, <, \{G_x\}_{x \in V(T)})$ be a distinguishing delayed decomposition of an ordered graph $(G, <)$. If x is a node of T with at least 3 children, there are no consecutive children y_1, y_2 of x both labelled with O.*

Proof. Let x be such a node, and y_1, y_2 be consecutive children of x. Since the delayed decomposition $(T, <, \{G_x\}_{x \in V(T)})$ is distinguishing, there is some vertex $v \in V(G) \setminus L(x)$ which is adjacent to all vertices in one of $L(y_1), L(y_2)$ and none in the other. Let u_1 be any vertex in $L(y_1)$ and u_2 any vertex in $L(y_2)$. Since $v \notin L(x)$ then u_1 and v have the same last common ancestor z as u_2 and v, and z is a proper ancestor of x. Suppose by contradiction that z is a proper ancestor of $p(x)$. Then, the grandchild of z which is an ancestor of u_1 is also the grandchild of z which is an ancestor of u_2, call it x'. Let v' be the ancestor of v which is a grandchild of z. If $x'v' \in E(G_z)$ then $u_1v \in E(G)$ and $u_2v \in E(G)$, a contradiction. Similarly, if $x'v' \notin E(G_z)$ then $u_1v \notin E(G)$ and $u_2v \notin E(G)$, again a contradiction. Therefore, z is the parent of x. Let v' be be the ancestor of v which is a grandchild of z. Then, v' is a cousin of y_1 and y_2. If $i \in \{1, 2\}$ is such that v is adjacent to all vertices in $L(y_i)$ then $v'y_i \in E(G_{p^2(y_i)})$ so one of y_1 and y_2 is not labelled with O. ◀

In view of Lemma 13, if x is a node of T with at least 3 children, and y is a child of x labelled O which is not the first child of x, then the predecessor y' of y has either label L or label R. If y' has label L, we refine the label of y to O_L and if y' has label R, we refine the label of y to O_R.

Given a node x of a delayed structured tree, if y is a vertex of G_x whose parent is labelled with R then there exists some vertex $y' > V(G_x)$ which is adjacent to y. This observation will be our main tool to prove that graphs with large delayed rank have large complete interval minors (see Theorem 19). For this reason, we define the *type* of a node $x \in V(T)$ as the label of its *parent* in T.

The *delayed rank* of an ordered graph $(G, <)$ is defined as follows:

- If $(G, <)$ is monotone bipartite, $(G, <)$ has delayed rank 0,
- Otherwise, we compute the distinguishing delayed decomposition of $(G, <)$. For each quotient graph $(G_x, <)$, if $(G_x, <)$ is monotone bipartite we say that $(G_x, <)$ is a *refined quotient graph* of $(G, <)$. Otherwise, note that x has at least 3 children (since there are only edges between cousins in G_x). In that case, if the first child of x is labelled with O, we remove all its children from G_x. Then, we partition the vertices into types R, L, O_R and O_L. Set $R' = \{R, O_R\}$ and $L' = \{L, O_L\}$. We partition the edges into four types, depending on whether the type of their left endpoint is in R' or L', and similarly for their right endpoint, denote these four types by $R'R', R'L', L'R'$ and $L'L'$. The *refined quotient graphs* of $(G_x, <)$ are then defined as follows:
  - The graph induced by the edges $R'R'$, to which we remove all vertices before the first vertex of type R and after the last vertex of type R.
  - The graph induced by the edges $R'L'$, to which we remove all vertices before the first vertex of type L and after the last vertex of type L.
  - The graph induced by the edges $L'R'$, to which we remove all vertices before the first vertex of type R and after the last vertex of type R.
  - The graph induced by the edges $L'L'$, to which we remove all vertices before the first vertex of type L and after the last vertex of type L.

  Then, the delayed rank of $(G, <)$ is 1 more than the maximum delayed rank of all its refined quotient graphs.

Observe first that siblings form an independent set in G_x, so removing all the children of the first child of x is just removing an independent set, at the beginning of the order of G_x. Similarly, the vertices before the first vertex of type R are at the beginning of the order of G_x and all have type either L or O_L, so they cannot induce any edge of type $R'R'$ or $L'R'$. As for the vertices after the last vertex of type R, they are at the end of the order of G_x and

all have type either L, O_L or O_R, and in the latter case they are siblings and come before the vertices of type L and O_L. Thus, all these vertices cannot induce any edge of type $R'R'$ or $L'R'$. Similar observations hold for the vertices before the first vertex of type L and after the last vertex of type L.

We now give some simple properties of the delayed rank. The first one gives a simple way of computing the delayed rank of an ordered graph. We first need some definitions.

If G is an ordered graph, we define a sequence $(\mathcal{G}_i(G))_{i\geq 0}$ of sets of graphs as follows. Set $\mathcal{G}_0(G) = \{G\}$. Suppose that $\mathcal{G}_i(G)$ is defined. Then, for every $H \in \mathcal{G}_i(G)$, add all the refined quotient graphs of H to $\mathcal{G}_{i+1}(G)$.

The following results are immediate from the definitions, and can be proved easily by induction on r.

▶ **Lemma 14.** *An ordered graph G has delayed rank at least r if and only if $\mathcal{G}_r(G) \neq \emptyset$.*

▶ **Lemma 15.** *If G, H are ordered graphs such that $H \in \mathcal{G}_r(G)$ for some $r \geq 0$ then H is a subgraph of G.*

The following result bounds the size of each $\mathcal{G}_r(G)$. It will be important in the analysis of the running time of the algorithm of Theorem 4.

▶ **Lemma 16.** *If G is an ordered graph with n vertices and m edges, for every $r \geq 1$, the total number of edges of all graphs in $\mathcal{G}_r(G)$ is at most m. Thus, for every $r \geq 1$, there are at most $4mn$ graphs in $\mathcal{G}_r(G)$.*

Proof. We prove the first property by induction on r. For $r = 1$, Remark 9 implies that the total number of edges of all quotient graphs of G is at most m. Since the refined quotient graphs of G are obtained from the quotient graphs by partitioning the edges and removing some edges, the total number of edges of all graphs in $\mathcal{G}_1(G)$ is at most m. Suppose now that the property holds for some $r \geq 1$. By definition, $\mathcal{G}_{r+1}(G) = \bigcup_{H\in\mathcal{G}_r(G)} \mathcal{G}_1(H)$. By the induction hypothesis at rank 1, if H has m' edges then the total number of edges of all graphs in $\mathcal{G}_1(H)$ is at most m'. By the induction hypothesis at rank r, the total number of edges over all $H \in \mathcal{G}_r(G)$ is at most m. Thus, the total number of edges of all graphs in $\mathcal{G}_{r+1}(G)$ is at most m. This proves the first part of the statement.

If $m = 0$ then G is monotone bipartite so $\mathcal{G}_1(G) = \emptyset$ and $|\mathcal{G}_1(G)| \leq 4mn$. Otherwise, by Remark 9, G has at most $n - 1$ quotient graphs, each of which gives rise to at most 4 refined quotient graphs, so $|\mathcal{G}_1(G)| \leq 4(n - 1) \leq 4mn$. Now, suppose $r > 1$. By the first part of the statement, there are at most m graphs $H \in \mathcal{G}_{r-1}(G)$ with at least one edge. All other graphs in $\mathcal{G}_{r-1}(G)$ are monotone bipartite, hence they have no refined quotient graphs. If $H \in \mathcal{G}_{r-1}(G)$ then H has at most n vertices by Lemma 15 so, by Remark 9, H has at most n quotient graphs. Each of these quotient graphs gives rise to at most 4 refined quotient graphs, so $|\mathcal{G}_1(H)| \leq 4n$. Taking the union over all $H \in \mathcal{G}_{r-1}(G)$ which have at least one edge, we get $|\mathcal{G}_r(G)| \leq 4mn$. ◀

By Lemma 7, delayed substitution closure can alternatively be seen as substitution closure followed by edge union. Therefore, a simple inductive proof shows that the notion of delayed rank is simply a refinement of the notion of rank.

▶ **Lemma 17.** *If G has delayed rank r then G has rank at most $7r + 2$.*

3.2 Delayed rank and complete interval minors

We now prove the key result about graphs of large delayed rank: they contain large complete interval minors. The next lemma is the engine of our proof, it allows us to measure the progress we do in building the large complete interval minor when going from delayed rank r to delayed rank $r+1$.

We consider *looped* interval minors: an ordered graph $(H, <)$ (possibly with a loop on each vertex) with vertex set $v_1 < \ldots < v_h$ is an interval minor of an ordered graph $(G, <)$ if there exists a partition of $V(G)$ into intervals $I_1, \ldots, I_h$ such that whenever $v_i v_j \in E(H)$, there is an edge in G between I_i and I_j. A *looped* K_t is an ordered clique of size t, with a loop on every vertex. A *left-lazy looped* K_t is an ordered clique of size t, with a loop on every vertex, except the first one. A *right-lazy looped* K_t is an ordered clique of size t, with a loop on every vertex, except the last one. A *lazy looped* K_t is an ordered clique of size t, with a loop on every vertex, except the first one and the last one. See Figure 7 for an illustration of all these graphs.

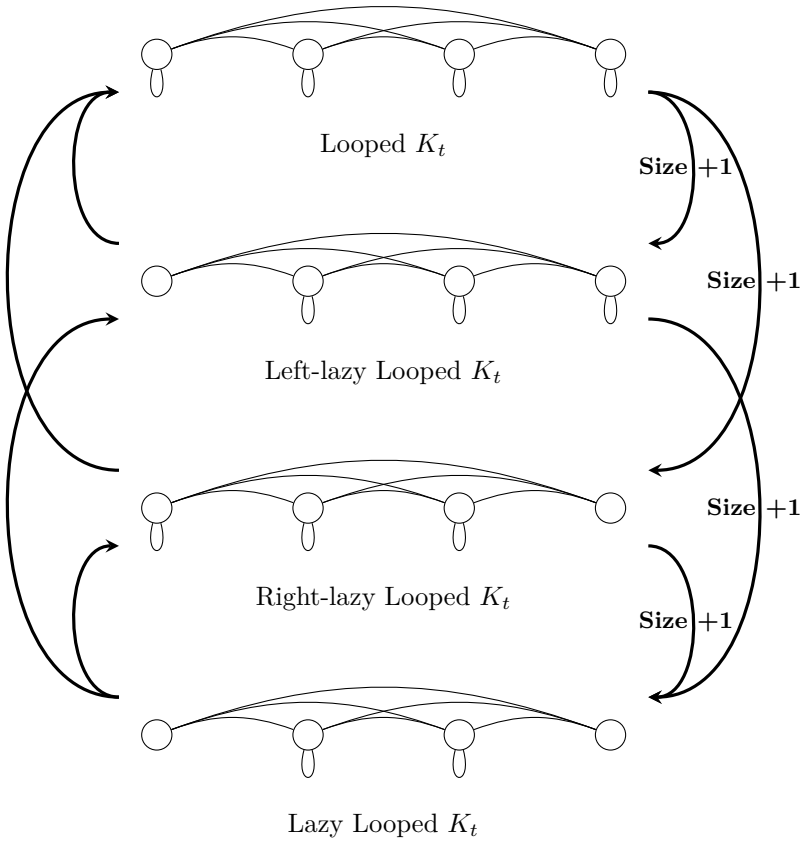

Figure 7 The various looped interval minors we consider. The arrows indicate the possible outcomes of Lemma 18.

▶ **Lemma 18.** *Let $r \geq 1$ and let $\mathcal{C}_r$ be the class of ordered graphs with delayed rank r. Then, the following assertions hold.*

1. *If every ordered graph in $\mathcal{C}_r$ contains a looped K_t interval minor, then every ordered graph in $\mathcal{C}_{r+1}$ contains a left-lazy or a right-lazy looped K_{t+1} interval minor.*
2. *If every ordered graph in $\mathcal{C}_r$ contains a left-lazy or a right-lazy looped K_t interval minor, then every ordered graph in $\mathcal{C}_{r+1}$ contains either a lazy looped K_{t+1} interval minor or a looped K_t interval minor.*
3. *If every ordered graph in $\mathcal{C}_r$ contains a lazy looped K_t interval minor, then every ordered graph in $\mathcal{C}_{r+1}$ contains a left-lazy or a right-lazy looped K_t interval minor.*

Proof. Consider an ordered graph $(G, <) \in \mathcal{C}_{r+1}$. Compute the delayed decomposition of $(G, <)$. By definition, there is a refined quotient graph H of G some type ($R'R'$, $R'L'$, $L'R'$ or $L'L'$) which is in $\mathcal{C}_r$. We consider several cases depending on the type of H.

- If H has type $R'R'$ then H is the graph induced by the edges $R'R'$, to which we remove all vertices before the first vertex of type R and after the last vertex of type R. Suppose that H contains a lazy looped K_t interval minor, and let $I_1 < \ldots < I_t$ be the intervals corresponding to this interval minor. Recall that $(I_1, \ldots, I_t)$ is a partition of $V(H)$. We argue that each interval I_j contains a vertex of type R.
  Since the first and the last vertex of H are vertices of type R, this is true for I_1 and I_t. Consider now any I_j with $1 < j < t$. Since the K_t interval minor is looped, there is an edge between two vertices of I_j, which means that there are two vertices in I_j which are of type R' but don't have the same parent (since siblings form an independent set in the quotient graphs). Therefore, there is a vertex of type R in I_j. Let $I_{t+1} = \{x \in V(G) : V(H) < x\}$. If $y_j \in I_j$ is a vertex of type R, it follows from the definition of the type R that y_j has a neighbor in I_{t+1}.
  Using I_{t+1}, we can then extend any looped K_t to a right lazy looped K_{t+1}, any left-lazy looped K_t to a lazy looped K_{t+1}, any right-lazy looped K_t to a looped K_t and any lazy looped K_t to a right-lazy looped K_t.
- If H has type $R'L'$ then H is the graph induced by the edges $R'L'$, to which we remove all vertices before the first vertex of type L and after the last vertex of type L. Suppose that H contains a lazy looped K_t interval minor, and let $I_1 < \ldots < I_t$ be the intervals corresponding to this interval minor. Recall that $(I_1, \ldots, I_t)$ is a partition of $V(H)$. We argue that each interval I_j contains a vertex of type L.
  Since the first and the last vertex of H are vertices of type L, this is true for I_1 and I_t. Consider now any I_j with $1 < j < t$. Since the K_t interval minor is looped, there is an edge between two vertices of I_j, which means that there exist $u < v \in I_j$ such that the type of u is in $R' = \{R, O_R\}$, and the type of v is in $L' = \{L, O_L\}$. Thus, there exist consecutive vertices $u < v \in I_j$ such that the type of u is in R' and the type of v is in L'. This implies that the type of v is L. Therefore, there is a vertex with type L in I_j. Let $I_0 = \{x \in V(G) : x < V(H)\}$. If $y_j \in I_j$ is a vertex of type L, it follows from the definition of the type L that y_j has a neighbor in I_0.
  Using I_0, we can extend any looped K_t to a left lazy looped K_{t+1}, any left-lazy looped K_t to a looped K_t, any right-lazy looped K_t to a lazy looped K_{t+1} and any lazy looped K_t to a left-lazy looped K_t.
- The case $L'R'$ is similar to the case $R'L'$ (except that we prove that each I_j contains a vertex of type R), and the case $L'L'$ is similar to the case $R'R'$ (except that we prove that each I_j contains a vertex of type L).

The result then follows immediately since we can extend any lazy looped K_t interval minor of H as desired, and since $H \in \mathcal{C}_r$. ◀

We easily deduce the main result of this section from Lemma 18.

▶ **Theorem 19.** *Every ordered graph with delayed rank at least* $3r-2$ *contains a* K_r *interval minor.*

Proof. We prove by induction on r that every ordered graph with delayed rank at least $3r-2$ has a looped K_r interval minor. For $r=1$, if G has delayed rank at least 1 then G contains at least one edge so G has a looped K_1 interval minor. Suppose that the property holds for $3r-2$. By Lemma 18, every graph of delayed rank at least $3r-1$ contains a left-lazy or a right-lazy looped K_{t+1} interval minor. Applying Lemma 18 again, every graph of delayed rank at least $3r$ contains either a lazy looped K_{t+2} interval minor or a looped K_{t+1} interval minor. Using Lemma 18 once more, every graph of delayed rank at least $3r+1$ contains either a left-lazy or a right-lazy looped K_{t+2} interval minor, which itself contains a looped K_{t+1} interval minor. ◀

Now, Theorem 3 follows immediately from combining Theorem 19 with Lemma 17.

4 Approximating the complete interval minor number

In this section, we present an algorithm to approximate the size of a largest complete interval minor in an ordered graph G. In Section 4.1, we give some implementation details on parts of the algorithm. In Section 4.2, we give a Ramsey-type theorem in the context of interval minors, which is crucial for bounding the approximation factor of the algorithm. In Section 4.3, we describe and analyse the algorithm.

4.1 More algorithmic tools

By Theorem 8, the delayed decomposition of an explicit ordered graph with n vertices and m edges can be computed in time $O(n+m)$. From there, it is simple to compute the refined quotients in the same running time.

▶ **Lemma 20.** *There is an algorithm which, given as input an explicit ordered graph* G *with* n *vertices and* m *edges, computes all the refined quotients of* G *in time* $O(n+m)$*.*

Proof. First, we start by checking whether G is monotone bipartite. To do so, we iterate over all edges to find the largest left endpoint of an edge, call it ℓ, and the smallest right endpoint of an edge, call it r. If $\ell < r$ then G is monotone bipartite and has no refined quotients, so we return the empty set. Otherwise, G is not monotone bipartite.

In that case, we compute the delayed decomposition $(T, <, \{G_x\}_{x\in V(T)})$ of G in time $O(n+m)$ using Theorem 8, with all the G_x stored explicitly. Then, we compute the label of every node x (L, R or O) by looking at its neighborhood in the graph $G_{p^2(x)}$. Thus, in time $O(n+m)$, we can compute the label of all nodes $x \in V(T)$. From this, we can compute the type (L, R, O_L, O_R or O) of every node of T, in time $O(n+m)$ overall. Then, for each quotient graph G_x, we check whether it is monotone bipartite. If so, we add it to the set of refined quotient graphs. Otherwise, we continue. Using the same method as above, this can be done in time linear in the size of G_x, so in time $O(n+m)$ overall.

We then remove from each G_x the first vertex as long as it is of type O. Then, for every edge of every G_x, we can compute its type by looking at the types of its endpoints. Once this is done, it is simple to compute the graphs induced by each of the four edge types. Finally, removing all vertices up to the first vertex of some type and after the last vertex of some type can also be done in time $O(n+m)$ overall. Note that the refined quotients are all stored explicitly. ◀

The *total size* of a delayed structured tree $(T, <, \{G_x\}_{x \in V(T)})$ is $|V(T)| + \sum_{x \in V(T)} |E(G_x)|$. The next result follows from a simple top-down dynamic programming algorithm.

▶ **Lemma 21.** *There is an algorithm which, given as input a delayed structured tree $(T, <, \{G_x\}_{x \in V(T)})$ of total size s, decides in time $O(s)$ whether there is a h-heavy leaf in T.*

4.2 A Ramsey-type result for complete interval minors

Before we move on to the algorithm, we present a Ramsey-type result for complete interval minors, which will be crucial for the bound on the "approximation factor" of our algorithm. Ramsey's theorem states that in every red/blue coloring of the edges of K_n, there is a monochromatic clique of size $\log(n)/2$, and this bound is essentially tight up to constant factors. We consider the analog question in the context of interval minors for ordered graphs. A red/blue coloring of the edges of the ordered graph K_n contains a *red (resp. blue) complete interval minor* of size t if there exists a partition of $V(G)$ into t intervals such that there is a red (resp. blue) edge between any two of these intervals. A *monochromatic complete interval minor* of size t is a red or a blue complete interval minor of size t.

We now prove that the bounds are much better for monochromatic complete interval minors than for monochromatic cliques.

▶ **Lemma 22.** *For every red/blue coloring of the edges of the ordered graph K_n, there is a monochromatic complete interval minor of size $2^{\sqrt{\log(n)}-1}$.*

Proof. Suppose that there does not exist a red complete interval minor of size $2^{\sqrt{\log(n)}-1}$. We prove that for every $i \leq \sqrt{\log(n)}-1$, we can find 2^i intervals, each of size at least $n/2^{i\sqrt{\log(n)}}$, with only blue edges between any two of these intervals. The property is trivial for $i = 0$. Suppose that the property holds for some $i < \sqrt{\log(n)}-1$. Then, there exist 2^i intervals, each of size at least $n/2^{i\sqrt{\log(n)}}$, with only blue edges between any two of them. Consider one such interval I, and cut it into $2^{\sqrt{\log(n)}-1}$ subintervals, each of size at least $\lfloor |I|/2^{\sqrt{\log(n)}-1} \rfloor$. Then, each subinterval has size at least $\lfloor |I|/2^{\sqrt{\log(n)}-1} \rfloor \geq n/2^{(i+1)\sqrt{\log(n)}-1} - 1 \geq n/2^{(i+1)\sqrt{\log(n)}}$. Indeed:

$$\begin{aligned}
\frac{n}{2^{(i+1)\sqrt{\log(n)}-1}} - 1 \geq \frac{n}{2^{(i+1)\sqrt{\log(n)}}} &\iff \frac{2n - 2^{(i+1)\sqrt{\log(n)}}}{2^{(i+1)\sqrt{\log(n)}}} \geq \frac{n}{2^{(i+1)\sqrt{\log(n)}}} \\
&\iff 2n - 2^{(i+1)\sqrt{\log(n)}} \geq n \\
&\iff n \geq 2^{(i+1)\sqrt{\log(n)}} \\
&\iff \log(n) \geq (i+1)\sqrt{\log(n)} \\
&\iff i \leq \sqrt{\log(n)} - 1.
\end{aligned}$$

Since there does not exist a red complete interval minor of size $2^{\sqrt{\log(n)}-1}$, there are two subintervals with no red edge between them, hence only blue edges between them. Doing this in each of the 2^i intervals, we find 2^{i+1} subintervals, each of size at least $n/2^{(i+1)\sqrt{\log(n)}}$, with only blue edges between any two of them. This result for $i = \sqrt{\log(n)} - 1$ proves the existence of a blue complete interval minor of size $2^{\sqrt{\log(n)}-1}$. ◀

The previous bound is almost sharp. To see it, set $q = 2^{\sqrt{\log(n)\cdot\log\log(n)}}$ and consider a random red/blue edge coloring of the edges of the ordered graph K_q. Then, taking lexicographic products of this graph, it can be shown that for n large enough, with high probability, this yields a red/blue coloring of the edges of the ordered graph K_n where the largest monochromatic complete interval minor has size $2^{2\cdot\sqrt{\log(n)\cdot\log\log(n)}}$.

4.3 The algorithm

We now move to the description of the algorithm. We will need the following technical lemma to bound the "approximation factor" of our algorithm.

▶ **Lemma 23.** *For every $t \geq 1$, there exists a function $h_t : \{0, 1, \ldots, 3t-2\} \to \mathbb{R}^+$, such that the following holds:*

1. *$f : t \mapsto h_t(0)$ satisfies $f(t) = 2^{2^{2^{O(t)}}}$,*
2. *For every $r \in [3t-2]$, $h_t(r) \leq 2^{\sqrt{\sqrt{\log\left((h_t(r-1)/2)^{1/(2t-1)}-4\right)}-1}-1} - 4$,*
3. *For every $r \in \{0, \ldots, 3t-2\}$, $h_t(r) \geq 4$.*

Proof. For every integer $t \geq 1$ and every integer $r \in \{0, \ldots, 3t-2\}$, set

$$g_t(r) = 4^{3t-2-r} + \frac{4^{3t-2-r}-1}{3}\log(512t)$$

and $h_t(r) = 2^{2^{g(r)}}$. It can be verified that the functions h_t have the desired properties. ◀

We are now ready to prove Theorem 4. In the following statement, we assume that the ordered graph $(G, <)$ is given explicitly.

▶ **Theorem 4.** *There is a triply exponential function f and a decision algorithm which, given as input an ordered graph $(G, <)$ with n vertices and m edges and an integer t, satisfies the following:*

- *If the algorithm returns Yes then $(G, <)$ contains K_t as an interval minor.*
- *If the algorithm returns No then $(G, <)$ does not contain $K_{f(t)}$ as an interval minor.*
- *The algorithm runs in time $O(t \cdot mn^2)$.*

Furthermore, when the algorithm returns Yes, it can also output a model of a K_t interval minor within the same running time.

Proof. The high-level description of the algorithm is extremely simple: we compute the delayed rank of $(G, <)$. If this rank is at least $3t-2$, we return Yes. Otherwise, we look whether there is a $(2t-3)$-heavy leaf in one of the delayed structured trees that we computed. If so, we return Yes. Otherwise, we return No.

More formally, we compute the sets $\mathcal{G}_0(G), \ldots, \mathcal{G}_{3t-2}(G)$. If $\mathcal{G}_{3t-2}(G) \neq \emptyset$, we return Yes. Otherwise, if there exists some $H \in \mathcal{G}_0(G) \cup \ldots \cup \mathcal{G}_{3t-2}(G)$ whose delayed decomposition tree contains a $(2t-3)$-heavy leaf then we return Yes. Otherwise, we return No.

For every $t \geq 1$, let h_t be the function provided by Lemma 23. Then, define $f : t \mapsto h_t(0)$, and note that $f(t) = 2^{2^{2^{O(t)}}}$.

▷ Claim. If the algorithm returns Yes then $(G, <)$ has a K_t interval minor.

Proof of the Claim. By Lemma 14, if $\mathcal{G}_{3t-2}(G) \neq \emptyset$ then G has delayed rank at least $3t-2$ so Theorem 19 implies that G has a K_t interval minor. If there exists some $H \in \mathcal{G}_0(G) \cup \ldots \cup \mathcal{G}_{3t-2}(G)$ whose delayed decomposition tree contains a $(2t-3)$-heavy

leaf then by Lemma 15, H is a subgraph of G and H is the realization of a delayed structured tree that contains a $(2t-3)$-heavy leaf so by Lemma 12, H contains a clique of size t. Therefore, G itself contains a clique of size t, hence a K_t interval minor. ◁

▷ Claim. If G has a $K_{f(t)}$ interval minor then the algorithm returns Yes.

Proof of the Claim. Suppose that there is no graph $H \in \mathcal{G}_0(G) \cup \ldots \cup \mathcal{G}_{3t-2}(G)$ whose delayed decomposition tree contains a $(2t-3)$-heavy leaf (otherwise we return Yes). We prove by induction that for every $0 \leq r \leq 3t-2$, there exists a graph $H_r \in \mathcal{G}_r(G)$ which has a complete interval minor of size $h_t(r)$. In particular, this will imply that $\mathcal{G}_{3t-2}(G) \neq \emptyset$, so the algorithm returns Yes. For $r = 0$, set $H_0 = G \in \mathcal{G}_0(G)$, which has a complete interval minor of size $f(t) = h_t(0)$ by assumption. Suppose that the property holds for some $r \in \{0, 1, \ldots, 3t-3\}$. Consider an interval family $\mathcal{I}$ which realizes the complete interval minor of size $h_t(r)$ in H_r. In particular, since $h_t(r) \geq 4$ then H_r is not bipartite so the refined quotient graphs of H_r are in $\mathcal{G}_{r+1}(G)$. Let $(T, <, \{G_x\}_{x \in V(T)})$ be the delayed decomposition of H_r. Set $b_r = (h_t(r)/2)^{1/(2t-1)} - 2$, so that $h_t(r) = 2(b_r+2)^{2t-1}$. By Lemma 11, since H_r doesn't contain a $(2t-3)-$heavy leaf, there is no $(2t-1)$-interval path in H_r. Then, Lemma 10 implies that there is a b_r-branching node x in T, which means that the corresponding quotient graph G_x has a complete interval minor of size b_r. After possibly removing the first vertices of type O, the resulting subgraph of G_x still has a complete interval minor of size $b_r - 2$ since we only removed an independent set. Applying Lemma 22 twice, we get that one of the types $R'R', R'L', L'R'$ and $L'R'$ satisfies that the subgraph induced by the edges of this type has a complete interval minor of size at least $2^{\sqrt{\sqrt{\log(b_r-2)}-1}-1}$. Removing the first vertices and the last vertices, which both form a stable set, decreases the size of the complete interval minor by at most 4, so one of the refined quotient graphs H_{r+1} of H_r has a complete interval minor of size at least $2^{\sqrt{\sqrt{\log(b_r-2)}-1}-1} - 4 = 2^{\sqrt{\sqrt{\log\left((h_t(r)/2)^{1/2t-1}-4\right)}-1}-1} - 4 \geq h_t(r+1)$ by definition of h_t. Thus, $H_{r+1} \in \mathcal{G}_{r+1}(G)$ has a complete interval minor of size at least $h_t(r+1)$, as desired. ◁

▷ Claim. This algorithm can be implemented to run in time $O(t \cdot mn^2)$.

Proof of the Claim. By Lemma 20, the set of refined quotient graphs (stored explicitly) of an explicit ordered graph with v vertices and e edges can be computed in time $O(v+e)$. Thus, $\mathcal{G}_1(G)$ can be computed in time $O(n+m)$, with all graphs being stored explicitly.

Suppose that we already computed $\mathcal{G}_r(G)$ for some $1 \leq r < 3t-2$, with all graphs being stored explicitly. Then, Lemmas 15 and 16 imply that it contains at most $4mn$ graphs, each with at most n vertices, and with at most m edges in total over all graphs in $\mathcal{G}_r(G)$. Thus, $\mathcal{G}_{r+1}(G)$ can be computed in time $\sum_{H \in \mathcal{G}_r(G)} O(|V(H)|+|E(H)|) = O(4mn \cdot n + m) = O(mn^2)$, with all graphs being stored explicitly. Therefore, computing $\mathcal{G}_0(G) \cup \ldots \cup \mathcal{G}_{3t-2}(G)$ can be done in time $O(t \cdot mn^2)$.

Finally, given an explicit ordered graph with v vertices and e edges, its delayed decomposition can be computed in time $O(v+e)$, hence the corresponding delayed structured tree has total size $O(v+e)$. Then, by Lemma 21, the algorithm can compute in time $O(v+e)$ whether there is a $(2t-3)$-heavy leaf in it. Thus, by iterating over all graphs in $\mathcal{G}_0(G) \cup \ldots \cup \mathcal{G}_{3t-2}(G)$, the algorithm can check whether there exists some $H \in \mathcal{G}_0(G) \cup \ldots \cup \mathcal{G}_{3t-2}(G)$ whose delayed structured tree contains a $(2t-3)$-heavy leaf. Since by Remark 9 there are $O(t \cdot mn)$ such graphs and together they contain at most $O(t \cdot m)$ edges, this can be done in time

$$\sum_{H \in \mathcal{G}_0(G) \cup \ldots \cup \mathcal{G}_{3t-2}(G)} O(|V(H)| + |E(H)|) = O(t \cdot mn \cdot n + t \cdot m) = O(t \cdot mn^2). \quad ◁$$

For the last statement, observe that the proofs of Lemmas 12 and 18 and Theorem 19 are all algorithmic and can be implemented efficiently. Note also that in the course of the algorithm, we compute all the graphs in $\mathcal{G}_0(G) \cup \ldots \cup \mathcal{G}_{3r-2}(G)$ and their delayed decompositions. ◀

References

1 Noga Alon, Andrzej Lingas, and Martin Wahlen. Approximating the maximum clique minor and some subgraph homeomorphism problems. *Theoretical Computer Science*, 374(1):149–158, 2007. doi:10.1016/j.tcs.2006.12.021.

2 Maria Axenovich, Jonathan Rollin, and Torsten Ueckerdt. Chromatic number of ordered graphs with forbidden ordered subgraphs. *Combinatorica*, 38(5):1021–1043, October 2018. doi:10.1007/s00493-017-3593-0.

3 Édouard Bonnet, Romain Bourneuf, Colin Geniet, and Stéphan Thomassé. Factoring pattern-free permutations into separable ones. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 752–779, 2024. doi:10.1137/1.9781611977912.30.

4 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: Small classes. *Combinatorial Theory*, June 2022. doi:10.5070/C62257876.

5 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Toruńczyk. Twin-width IV: Ordered graphs and matrices. *Journal of the ACM*, 71(3), June 2024. doi:10.1145/3651151.

6 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: Tractable FO model checking. *Journal of the ACM*, 69(1), November 2021. doi:10.1145/3486655.

7 Romain Bourneuf and Stéphan Thomassé. Bounded twin-width graphs are polynomially χ-bounded. *Advances in Combinatorics*, 2025(2):19p, 2025. doi:10.19086/aic.2025.2.

8 Marcin Briański, James Davies, and Bartosz Walczak. Colouring graphs without an induced ordered matching, In preparation.

9 Maria Chudnovsky, Irena Penev, Alex Scott, and Nicolas Trotignon. Substitution and χ-boundedness. *Journal of Combinatorial Theory, Series B*, 103(5):567–586, 2013. doi:10.1016/j.jctb.2013.02.004.

10 Josef Cibulka and Jan Kynčl. Better upper bounds on the Füredi-Hajnal limits of permutations. In *Proceedings of the 2017 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2280–2293, 2017. doi:10.1137/1.9781611974782.150.

11 David Eppstein. Finding large clique minors is hard. *Journal of Graph Algorithms and Applications*, 13(2):197–204, February 2009. doi:10.7155/jgaa.00183.

12 Jacob Fox. Stanley-Wilf limits are typically exponential, 2013. arXiv:1310.8378.

13 Zoltán Füredi and Péter Hajnal. Davenport-Schinzel theory of matrices. *Discrete Mathematics*, 103(3):233–251, 1992. doi:10.1016/0012-365X(92)90316-8.

14 Tibor Gallai. Transitiv orientierbare graphen. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(1):25–66, March 1967. doi:10.1007/BF02020961.

15 Jesse T. Geneson and Peter M. Tian. Extremal functions of forbidden multidimensional matrices. *Discrete Mathematics*, 340(12):2769–2781, 2017. doi:10.1016/j.disc.2017.08.017.

16 Maximilian Gorsky, Michał T. Seweryn, and Sebastian Wiederrecht. Polynomial bounds for the graph minor structure theorem, 2025. doi:10.48550/arXiv.2504.02532.

17 Sylvain Guillemot and Daniel Marx. Finding small patterns in permutations in linear time. In *Proceedings of the 2014 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 82–101, 2014. doi:10.1137/1.9781611973402.7.

18 Sepehr Hajebi, Yanjia Li, and Sophie Spirkl. List-3-coloring ordered graphs with a forbidden induced subgraph. *SIAM Journal on Discrete Mathematics*, 38(1):1158–1190, 2024. doi:10.1137/22M1515768.

19 Ken ichi Kawarabayashi, Yusuke Kobayashi, and Bruce Reed. The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B*, 102(2):424–435, 2012. doi:10.1016/j.jctb.2011.07.004.

20 Vít Jelínek and Stanislav Kučera. On the structure of matrices avoiding interval-minor patterns. *Advances in Applied Mathematics*, 101:70–99, 2018. doi:10.1016/j.aam.2018.07.005.

21 Martin Klazar and Adam Marcus. Extensions of the linear bound in the Füredi–Hajnal conjecture. *Advances in Applied Mathematics*, 38(2):258–266, 2007. doi:10.1016/j.aam.2006.05.002.

22 Tuukka Korhonen, Michał Pilipczuk, and Giannos Stamoulis. Minor containment and disjoint paths in almost-linear time. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 53–61, Los Alamitos, CA, USA, October 2024. IEEE Computer Society. doi:10.1109/FOCS61266.2024.00014.

23 Yaping Mao, Hongjian Lai, Zhao Wang, and Zhiwei Guo. Interval minors of complete multipartite graphs, 2015. arXiv:1508.01263.

24 Adam Marcus and Gábor Tardos. Excluded permutation matrices and the Stanley–Wilf conjecture. *Journal of Combinatorial Theory, Series A*, 107(1):153–160, 2004. doi:10.1016/j.jcta.2004.04.002.

25 Bojan Mohar, Arash Rafiey, Behruz Tayfeh-Rezaie, and Hehui Wu. Interval minors of complete bipartite graphs. *Journal of Graph Theory*, 82(3):312–321, 2016. doi:10.1002/jgt.21903.

26 Michał Pilipczuk and Marek Sokołowski. Graphs of bounded twin-width are quasi-polynomially χ-bounded. *Journal of Combinatorial Theory, Series B*, 161:382–406, 2023. doi:10.1016/j.jctb.2023.02.006.

27 N. Robertson and Paul D. Seymour. Graph Minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.

28 Neil Robertson and Paul D. Seymour. Graph Minors. XVI. Excluding a non-planar graph. *Journal of Combinatorial Theory, Series B*, 89(1):43–76, 2003. doi:10.1016/S0095-8956(03)00042-X.

29 Martin Wahlén. On the complexity of approximating the Hadwiger number. *Theoretical Computer Science*, 410(8):994–996, 2009. doi:10.1016/j.tcs.2008.12.020.