Upper Clique Transversal on Interval Graphs and Beyond

Lars Jaffke
Norwegian School of Economics, Bergen, Norway

Paloma de Lima
Norwegian School of Economics, Bergen, Norway
IT University of Copenhagen, Denmark

Amir Nikabadi
IT University of Copenhagen, Denmark

Abstract

The Upper Clique Transversal (UCT) problem asks for the size of the largest minimal set of vertices intersecting all the maximal cliques of the input graph. This problem was recently introduced by Milanič and Uno [WG 2023], who studied its complexity on several graph classes. They showed that the problem is NP-hard on chordal graphs, and gave polynomial-time algorithms for UCT on split and on proper interval graphs. They left open the complexity of UCT on interval graphs. In this work we settle this question by giving a polynomial-time algorithm for UCT on interval graphs. We show that even on the more general class of rooted directed path graphs, which can be understood as a "tree-like version" of interval graphs, the problem remains polynomial-time solvable. On the negative side, we observe as consequences of the NP-hardness proof for chordal graphs due to Milanič and Uno that the problem is NP-hard on graphs of path-independence number two (interval graphs have path-independence number one) and on well-partitioned chordal graphs which lie between split and chordal graphs.

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

Keywords and phrases interval graphs, rooted directed path graphs, clique transversal

Digital Object Identifier 10.4230/LIPIcs.MFCS.2026.99

Funding de Lima and Nikabadi acknowledge the support of the Independent Research Fund Denmark grant agreement number 2098-00012B.

1 Introduction

Graph optimization problems in which the goal is to compute a small vertex or edge subset satisfying a given property are the core of algorithmic graph theory. Although such problems are typically NP-hard, they often admit naïve polynomial-time algorithms that compute a *minimal* solution, as opposed to a minimum one.

For example, in the case of Vertex Cover, one can start with a set S containing all vertices of the graph and iteratively remove vertices whose neighbors are all still in S. To analyze the worst-case behavior of these naïve algorithms, one can ask for the maximum size of a minimal solution, which serves as an upper bound on the size of the output these algorithms may produce. This motivates the study of "upper" (also called MaxMin) variants of classical algorithmic problems.

Given a problem Π with the task of computing a minimum subset of vertices (or edges) satisfying property π, the problem Upper Π takes as input a graph G and an integer k and the question is whether G has *minimal* subset of vertices of size at least k satisfying π. Upper variants of several classical problems have been considered in the literature, including, for instance, vertex cover [8, 13, 24], feedback vertex set [14, 20], domination [1, 4, 18], edge

domination [23] and edge cover [19]. Motivated by this initial question the study of upper variants of optimization problems grew into a field of its own, often unveiling new insights into the structure of the original problems.

Very recently, Milanič and Uno [21] considered the upper variant of computing minimum clique transversals. A *clique transversal* is a set of vertices that intersects all maximal cliques of a graph. In the Upper Clique Transversal (UCT) problem, the task is to compute the maximum size of a minimal clique transversal of a graph. Note that this problem is of a harder nature: even though the problem of computing a minimum cardinality clique transversal has been extensively studied in the literature [3, 9, 15], there is no naïve algorithm that computes a minimal solution on general graphs. In their work, Milanič and Uno show UCT is NP-complete on chordal graphs, chordal bipartite graphs, and line graphs of bipartite graphs, but solvable in linear time on the classes of split graphs, bounded clique-width graphs and proper interval graphs. They leave as an open problem whether UCT is polynomial-time solvable on interval graphs.

We answer the main algorithmic open question left by Milanič and Uno [21] by providing a polynomial-time algorithm for UCT on interval graphs.

▶ **Theorem 1.** *There is an algorithm that solves Upper Clique Transversal on n-vertex interval graphs in $\mathcal{O}(n^3)$ time.*

It is worth to mention that our approach is completely different from that of Milanič and Uno [21] for proper interval graphs. Their algorithm relies on a linear-time algorithm for the maximum induced matching problem in bipartite permutation graphs due to Chang [10]. They first show that the upper clique transversal number of a graph G is at most the induced matching number of a bipartite graph constructed from G (the so called vertex-clique incidence graph). For the case of unit interval graphs, they show that this bipartite graph is in fact a bipartite permutation graph, which allows them to use the algorithm of Chang to compute an upper clique transversal of the input graph.

Next, we consider the class of rooted directed path graphs [22], which are intersection graphs of directed paths in a rooted directed tree T. Note that interval graphs form a subclass of rooted directed path graphs when T is a path. We show that also on this class, UCT is polynomial-time solvable.

▶ **Theorem 2.** *There is an algorithm that solves Upper Clique Transversal on n-vertex rooted directed path graphs in $\mathcal{O}(n^5)$ time.*

Our algorithms use dynamic programming on a corresponding representation of the input graph. One of the main difficulties imposed by the nature of UCT to the dynamic programming approach is that it does not seem to suffice to consider only minimal sets as partial solutions. At the same time, we have to ensure that the solution output by the algorithm in the end is minimal. Using the structure of the graph classes we consider we can keep partial solutions very close to being minimal, and keep track of how that property evolves over the course of the algorithm. Our approach is self-contained and holds the potential to be generalizable to other super-classes of interval graphs, such as graphs of bounded simultaneous interval number, a graph parameter recently introduced by Beisegel et al. [5] based on a generalization of the interval representation of interval graphs.

Finally, we observe UCT remains NP-hard on graphs of path-independence number two, implying that the polynomial-time algorithm of Theorem 1 cannot be generalized to graphs of bounded path-independence number. The path-independence number of a graph was defined by Beisegel et al. [5] analogously to the tree-independence number introduced by

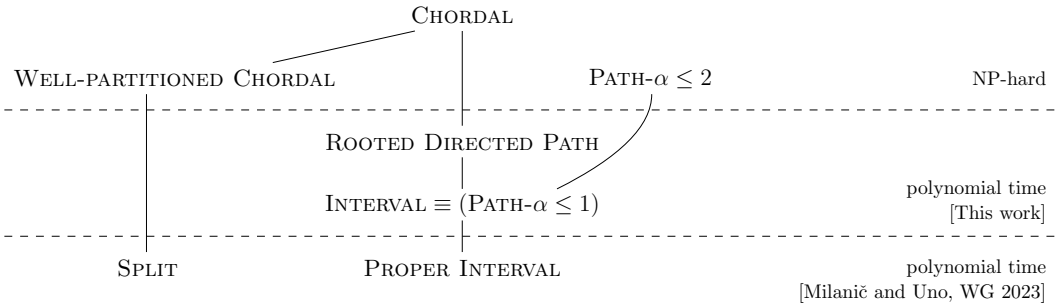

Figure 1 Overview of the results.

Dallard et al. [12]. Interval graphs are the graphs with path-independence number one. The NP-hardness of UCT on graphs of path-independence number two is obtained by the observation that the graphs in the reduction of Milanič and Uno [21] for chordal graphs are also graphs of path-independence number two. We also point out these graphs are well-partitioned chordal graphs, a class defined by Ahn et al. [2] that sits in-between split graphs (where the problem is polynomial-time solvable [21]) and chordal graphs. Our results are summarized in Figure 1.

The remainder of the paper is organized as follows. In Section 2, we state some definitions and the observations regarding the hardness results. We prove Theorem 1 in Section 3. We conclude with some open problems related to our results in Section 5. Proofs of statements marked with "♣" can be found in the full version.

2 Preliminaries

We use $\mathbb{N}$ to denote the set of positive integers. We let $[n] := \{1, \ldots, n\}$ for every $n \in \mathbb{N}$. We denote by $[a, b]$ an interval whose domain will always be clear from the context. For a set Ω, we denote by 2^Ω the set of all subsets of Ω. For $S \subseteq \Omega$, we let $\overline{S} = \Omega \setminus S$. Given $\mathcal{S} \subseteq 2^\Omega$, a *hitting set* for $\mathcal{S}$ is a set $X \subseteq \Omega$ such that for all $S \in \mathcal{S}$, $X \cap S \neq \emptyset$. If $X \cap S \neq \emptyset$, we may say that X *hits* S. Let $\mathcal{I}$ be a family of sets $\{S_i\}_{i \in \mathcal{I}}$, we denote by $\dot{\bigcup}_{i \in \mathcal{I}} S_i$ the *disjoint union* of $\{S_i\}_{i \in \mathcal{I}}$.

We use the *random access model* where atomic operations on integers take constant time. The values of the integers considered in this paper do not exceed $\mathcal{O}(n)$, where n is the input size, so all computations described here can be implemented on a Turing machine with logarithmic overhead in the input size.

All graphs are finite and have no loops or parallel edges. Let $G = (V, E)$ be a graph. A *clique* is a set of pairwise adjacent vertices. A clique C is *maximal* if there is no $u \in V(G) \setminus C$ that is adjacent to all vertices of C. A *clique transversal* is a set of vertices that hits all maximal cliques of a graph. The *upper clique transversal number* of a graph G is the largest size of a minimal clique transversal in G. An *independent set* is a set of pairwise non-adjacent vertices. We denote by $\alpha(G)$ the *independence number* of G, that is, the maximum size of any independent set in G. For $X \subseteq V(G)$, we denote the subgraph of G induced by X as $G[X]$, that is $G[X] = (X, \{uv \colon u, v \in X \text{ and } uv \in E\})$. We denote by $N(X)$ vertices of $G \setminus X$ with a neighbor in X, and $N[X] = N(X) \cup X$.

A graph is *chordal* if it has no induced cycle of length at least four. An *interval intersection representation* for a graph G is a set $\mathcal{I} = \{I_v \mid v \in V(G)\}$ of intervals over the real line, such that $uv \in E(G)$ if and only if $I_u \cap I_v \neq \emptyset$. A graph is an *interval graph* if it admits an interval intersection representation. Interval graphs form a subclass of chordal graphs.

A graph G is a *rooted directed path graph* if there is a rooted directed tree T and a set $\{P_v \mid v \in V(G)\}$ of directed paths in T such that for all $u, v \in V(G)$, $uv \in E(G)$ if and only if $V(P_u) \cap V(P_v) \neq \emptyset$.

A *path decomposition* for a graph G is a sequence of subsets $X_1, X_2, \ldots, X_t$ of $V(G)$ such that $\cup_{i\in[t]} X_i = V(G)$, for each $uv \in E(G)$, there exists i such that $u, v \in X_i$ and, for every $v \in V(G)$, the sets containing v are consecutive in the sequence. The *path independence number* of G, denoted *path-*$\alpha(G)$, is the smallest integer $\ell \in \mathbb{N}$ such that G admits a path decomposition with $\alpha(G[X_i]) \leq \ell$ for all $i \in [t]$. Interval graphs are precisely the graphs of path independence number one, as they are the graphs that admit a path decomposition in which every X_i is a clique.

Hardness results (old and new)

In their work introducing the UCT problem, Milanič and Uno [21] showed the following.

▶ **Theorem 3** (Milanič and Uno [21]). *Upper Clique Transversal is NP-complete on the class of chordal graphs.*

We will now make two observations about the graphs obtained from the reduction of Theorem 3. First, we show that they have path-independence number two. This implies that, unless P=NP, the algorithm of Theorem 1 cannot be generalized for graphs of bounded path-independence number. Second, we show these graphs are well-partitioned chordal graphs, a graph class defined by Ahn et al. [2] that contains the class of split graphs, and is contained in the class of chordal graphs. This implies that the algorithm of Milanič and Uno [21] for split graphs cannot be generalized to this larger graph class.

Before we proceed with our observations, we describe how the graphs in the reduction of Theorem 3 are constructed. The starting problem for the reduction is the Spanning Star Forest problem. A *star* is a tree with a single vertex of degree greater than one. A *spanning star forest* of G is a spanning subgraph of G that is a vertex-disjoint union of stars. In the Spanning Star Forest problem, the input is a graph G and an integer t, and the question is whether G has a spanning star forest with at least t edges. Given the graph G with vertex set V and edge set E that is an instance of Spanning Star Forest, the following graph H is constructed as an instance for UCT. The vertex set of H is $V(H) = V \cup \{x^e, y^e \mid e \in E\}$. The edge set of H is described as follows. The vertices in V form a clique in H and for each $e = uv \in E$, x^e is adjacent to $u, v \in V$ and to y^e. See Figure 2.

▶ **Observation 4.** *Let H be an instance of* Upper Clique Transversal *constructed in the proof of Theorem 3. Then H has path-independence number two.*

Proof. We construct a path decomposition for H as follows. For each $e \in E(G)$, we let $X_e = V \cup \{x^e, y^e\}$, and we consider the sets X_e with $e \in E(G)$ in an arbitrary order. Note that $\cup_{e\in E(G)} X_e = V(H)$ and for each edge of H there exists e' such that the edge is contained in $X_{e'}$. Moreover, since the vertices of V appear in all sets and, for each $e \in E(G)$, the vertices $\{x^e, y^e\}$ appear only in X_e, it holds that for every $v \in V(H)$, the sets containing v are consecutive in the sequence. Hence this is indeed a path decomposition of H. Now, note that $\alpha(H[X_e]) = 2$. Indeed, at most one vertex of V and at most one vertex from $\{x^e, y^e\}$ can be contained in an independent set of $H[X_e]$. ◀

For our next observation, we need the following definition.

A graph G is a *well-partitioned chordal* graph if there exists a tree $\mathcal{T}$ whose vertices are mapped to subsets of $V(G)$ such that the following holds. For simplicity, we use the same notation to refer to a node $X \in V(\mathcal{T})$ and the corresponding vertex set $X \subseteq V(G)$.

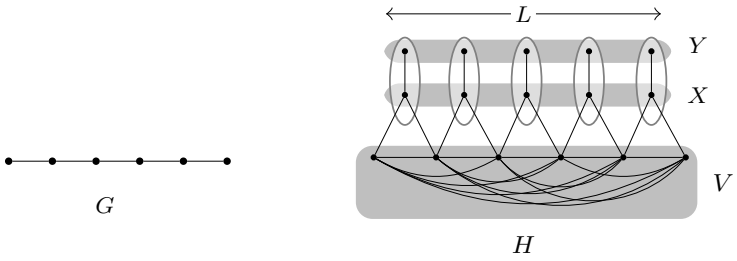

Figure 2 From left to right: a graph G and the graph H obtained from G.

(i) For each $X \in \mathcal{T}$ (called a *bubble*), $G[X]$ is a complete graph.
(ii) For each edge $XY \in E(\mathcal{T})$, there are subsets $X' \subseteq X$ and $Y' \subseteq Y$ such that $G[X' \cup Y']$ is a clique, and there are no other edges between X and Y in G.
(iii) For each pair of distinct $X, Y \in V(\mathcal{T})$ with $XY \notin E(\mathcal{T})$, there are no edges between X and Y in G.

▶ **Observation 5.** *Let H be an instance of* Upper Clique Transversal *constructed in the proof of Theorem 3. Then H is a well-partitioned chordal graph.*

Proof. Let $\mathcal{T}$ be the following star with a center $C = \{v \in H \mid v \in V\}$ and leaves L_e where $L_e = \{x^e, y^e\}$, for each $e \in E(G)$. First, observe that each vertex appears in exactly one set, and hence this is a partition of the vertex set of H. It follows, by construction, that each bubble is a clique, hence i holds. For ii, for an edge $CL_e \in E(\mathcal{T})$, take $X' = \{x^e\}$ and $Y' = \{u, v\} \subset C$, where $e = uv$. Note that $\{x^e, u, v\}$ is a triangle, and there are no other edges between C and L_e. Finally, for iii note that are no edges between L_e and $L_{e'}$ when $e \neq e'$. This concludes our claim that H is a well-partitioned chordal graph. ◀

3 Interval graphs

In this section, we give a polynomial-time algorithm for UCT on interval graphs. Given an interval representation $\{I_v = [\ell_v, r_v] \mid v \in V(G)\}$ of a graph G, we call $I_v = [\ell_v, r_v]$ the *interval of* v, refer to ℓ_v as its *left endpoint* and to r_v as its *right endpoint*.

▶ **Definition 6.** *An interval representation of a graph G is* canonical *if each point in $\cup_{v \in V(G)}\{\ell_v, r_v\}$ is a unique point in $[2n]$.*

We give a dynamic programming algorithm that uses the interval representation of the input interval graph. It is well-known that given an interval graph, such a representation can be computed in linear time [7], and it can be turned into a canonical one efficiently[1]. From now on, we assume that we have a canonical interval representation $\mathcal{R} = \{[\ell_v, r_v] \mid v \in V(G)\}$ of the input n-vertex graph G at hand. Before we proceed with the description of the algorithm, we need some auxiliary definitions.

▶ **Definition 7.** *For $i \in [2n]$, we let $V_i = \{v \in V(G) \mid \ell_v \leq i\}$, and we call $\mathsf{bd}(i) = \{v \in V(G) \mid \ell_v \leq i \wedge r_v \geq i\}$ the* boundary at i.

[1] In the worst case, this requires $\mathcal{O}(n \log n)$ time for sorting the endpoints of the intervals. This time is subsumed by the run time of the algorithm presented in this section.

Our algorithm traverses $\mathcal{R}$ in the natural way, that is, we have i going from 1 to $2n$; at each i, we compute information about partial solutions in $G[V_i]$. These partial solutions are subsets of V_i that intersect several, but not all, maximal cliques of $G[V_i]$. Observe that if a maximal clique C of $G[V_i]$ is contained in the boundary $\mathsf{bd}(i)$, then C might not be maximal in G as there can be vertices in $\overline{V_i}$ that are adjacent to all vertices in C. If, however, C has a vertex whose right endpoint is at most i, then no such vertex can exist, and C is in fact a maximal clique in G. Therefore, any clique transversal in G hits C. We call such maximal cliques C *forgotten at* i.

▶ **Definition 8.** *For $i \in [2n]$, we say that $S \subseteq V(G)$ is* forgotten at i *if* $\min_{v \in S} r_v \leq i$ *and* strictly forgotten at i *if* $\min_{v \in S} r_v < i$. *If i is clear from the context, we may simply call S* (strictly) forgotten. *We denote by $\mathcal{C}_i$ the set of maximal cliques of G that are forgotten at i.*

Partial solutions at i are minimal hitting sets of $\mathcal{C}_i$. Observe that all maximal cliques are forgotten at $2n$, so $\mathcal{C}_{2n}$ is the set of all maximal cliques of G, and partial solutions at $2n$ correspond to solutions in the entire graph.

▶ **Definition 9.** *Let H be a graph, $S \subseteq V(H)$, and $v \in S$. We say that v* has a witness w.r.t. S in H *if there is a maximal clique C in H such that $S \cap C = \{v\}$. We may drop "w.r.t. S" and/or "in H" whenever clear from the context.*

Note that a clique transversal is minimal if and only if each vertex in it has a witness.

▶ **Observation 10 (♣).** *Let C be a clique in G. Then, the interval of each vertex in C contains the point $\min_{v \in C} r_v$.*

▶ **Lemma 11.** *Let S be a minimal clique transversal of G. For any $i \in [2n]$, $|S \cap \mathsf{bd}(i)| \leq 2$. Furthermore, suppose $S \cap \mathsf{bd}(i) = \{x, y\}$ with $x \neq y$. Then, x has a witness strictly forgotten at i and y does not have a witness forgotten at i or vice versa.*

Proof. Suppose for a contradiction that $S \cap \mathsf{bd}(i) \supseteq \{x, y, z\}$. Note that the intervals of x, y, and z all contain the common point i, since $x, y, z \in \mathsf{bd}(i)$. We may assume that $\ell_x < \ell_y$ (up to renaming). Then, y has no witness forgotten at i: Suppose that C is a witness for y forgotten at i, and let $j = \min_{v \in C} r_v$. By Observation 10, j is in the interval of each vertex of C. But now, $\ell_x < \ell_y \leq j \leq i \leq r_x$ (where $\ell_y \leq j$ since $y \in C$), and therefore $j \in [\ell_x, r_x]$ and $x \in C$, a contradiction with C being a witness for y. Therefore, y must have a witness that is not yet forgotten at i. By reasoning similar to above, this allows us to infer that $r_y > r_x$ and $r_y > r_z$. Now, consider x and z. We may again assume (up to renaming) that $\ell_x < \ell_z$. Again, we have that z does not have a witness forgotten at i, so now let C be a witness for z. Then, $j = \min_{v \in C} r_v > i$. But now, $\ell_y \leq i < j \leq r_z < r_y$, therefore $y \in C$, a contradiction with C being a witness for z.

For the second claim, suppose that $S \cap \mathsf{bd}(i) = \{x, y\}$. We may assume that $r_x < r_y$, and we know that $i \leq r_x$. Let C be a witness for x and let $j = \min_{v \in C} r_v$. Then, the intervals of all vertices in C contain j by Observation 10. Suppose that $j \geq i$. Since $y \notin C$, this implies that $j > r_y$, but then $j > r_y > r_x$, and so $x \notin C$, a contradiction. We conclude that $j < i$, and therefore, C is strictly forgotten at i.

Now suppose that y also has a witness C' forgotten at i, and let $j' = \min_{v' \in C'} r_{v'}$; note that $j' \leq i$ and that $j \neq j'$. First suppose that $j' < j$. Since $x \notin C'$ and $r_x \geq i$, we have that $\ell_x > j'$. On the other hand, since $y \in C'$ and $r_y \geq i$, we have that $\ell_y \leq j'$. Similarly, $\ell_x \leq j$. Therefore, $\ell_y \leq j' < \ell_x \leq j \leq i \leq r_y$, meaning that $j \in [\ell_y, r_y]$ which implies $y \in C$, a contradiction. Now suppose that $j' > j$. Then, $\ell_x \leq j < j' \leq i \leq r_x$; therefore $j' \in [\ell_x, r_x]$, and so $x \in C'$ (Observation 10), a contradiction. ◀

▶ **Corollary 12.** *For any $i \in [2n]$, and $S \subseteq V(G)$, there is at most one vertex in $\mathsf{bd}(i)$ that has a witness w.r.t. S forgotten at i.*

Throughout the following, we let $\perp$ be a "blank" symbol. We let $\{\perp\} = \emptyset$.

▶ **Definition 13.** *For $i \in [2n]$, $v, w \in \mathsf{bd}(i) \cup \{\perp\}$, such that either $v \neq w$ or $v = w = \perp$, we let $c[i, v, w]$ be the largest size of any $S \subseteq V_i$ such that:*

1. *S is a hitting set of $\mathcal{C}_i$,*
2. *$S \cap \mathsf{bd}(i) = \{v, w\} \setminus \{\perp\}$,*
3. *each vertex $x \in S \setminus \{w\}$ has a witness that is forgotten at i, and*
4. *if $w \neq \perp$, then w does not have a witness w.r.t. S that is forgotten at i.*

We call any such set S a partial solution matching the index (i, v, w).

We now show how to compute the table entries as per Definition 13, and proceed inductively on i. For the base case, we let u denote the vertex such that $1 = \ell_u$; we let:

$$c[1, \perp, \perp] = 0,\ c[1, u, \perp] = -\infty,\ \text{and } c[1, \perp, u] = 1 \tag{1}$$

The first entry corresponds to the empty set which is a valid partial solution. Since $\mathcal{C}_1 = \emptyset$, u has no witness that is forgotten at 1. Therefore, in the second case, there is no valid partial solution matching the table index. In the third case, $\{u\}$ is a valid partial solution, as in this case, u is not required to have a witness yet.

Now suppose $i > 1$. We distinguish the following cases.

Case 1 ($i = \ell_u$ for some u) In this case, we have that $V_i = V_{i-1} \cup \{u\}$ and $\mathcal{C}_i = \mathcal{C}_{i-1}$. Let us first consider what happens when u is part of the table index. Since u is not part of any maximal clique that is forgotten at i, it cannot have a witness that is forgotten at i, therefore we have that $c[i, u, w] = -\infty$ for any $w \in \mathsf{bd}(i) \cup \{\perp\}$. Now suppose that $v \in \mathsf{bd}(i) \setminus \{u\}$ and let us consider the table entry $c[i, v, u]$. By Definition 13(3), in any solution matching that index, v has a witness forgotten at i. Moreover, since in this case $i = \ell_u$, such a witness also has to be forgotten at $i - 1$. Furthermore, by Lemma 11, such a solution could not have another vertex (other than u) that does not have a witness forgotten at i. Hence, at $i - 1$, all vertices of such a solution should have a witness forgotten at $i - 1$. So we let $c[i, v, u] = 1 + c[i - 1, v, \perp]$. In summary:

$$c[i, v, w] = \begin{cases} -\infty, & \text{if } v = u \\ 1 + c[i-1, v, \perp], & \text{if } v \in \mathsf{bd}(i) \text{ and } w = u \end{cases} \tag{2}$$

Note that we have not shown how to compute $c[i, \perp, u]$, as this (and the remaining cases) require further distinction.

Case 1.1 ($i - 1 = \ell_x$ for some x) In this case, we have that $\mathsf{bd}(i) = \mathsf{bd}(i-1) \dot\cup \{u\}$; therefore, we set $c[i, \perp, u] = 1 + c[i - 1, \perp, \perp]$ in analogy with above. In the remaining cases, we have that $v, w \in (\mathsf{bd}(i) \setminus \{u\}) \cup \{\perp\}$, and so $v, w \in \mathsf{bd}(i - 1)$. Therefore, we set $c[i, v, w] = c[i - 1, v, w]$.

$$c[i, v, w] = \begin{cases} 1 + c[i-1, \perp, \perp], & \text{if } v = \perp \text{ and } w = u \\ c[i-1, v, w], & \text{if } v, w \in (\mathsf{bd}(i) \setminus \{u\}) \cup \{\perp\} \end{cases} \tag{3}$$

Note that together with (2), the above covers all possibilities in Case 1.1.

Case 1.2 ($i - 1 = r_x$ for some x) In this case, $\mathsf{bd}(i) = \mathsf{bd}(i-1) \cup \{u\} \setminus \{x\}$, so we have to take the possibility that x was part of a partial solution into account. First, consider $c[i, \perp, u]$. A partial solution matching this index may or may not contain x; as in 1.1, such a partial solution cannot contain another vertex without a witness forgotten at $i - 1$.

We let $c[i, \perp, u] = 1 + \max\{c[i-1, x, \perp], c[i-1, \perp, \perp]\}$. Similarly, for any $w \neq u$, we have $w \in \mathsf{bd}(i-1) \cup \{\perp\}$ and let $c[i, \perp, w] = \max\{c[i-1, x, w], c[i-1, \perp, w]\}$. From now on, we may assume that $v \notin \{u, \perp\}$ and $w \neq u$; the former implies that $v \in \mathsf{bd}(i-1)$ and the latter that $w \in \mathsf{bd}(i-1) \cup \{\perp\}$. In this case, any partial solution at i that matches this table index has to have the same index at $i-1$, and so $c[i, v, w] = c[i-1, v, w]$.

$$c[i,v,w] = \begin{cases} 1 + \max\{c[i-1,x,\perp], c[i-1,\perp,\perp]\}, & \text{if } v = \perp \text{ and } w = u \\ \max\{c[i-1,x,w], c[i-1,\perp,w]\}, & \text{if } v = \perp \text{ and } w \neq u \\ c[i-1,v,w], & \text{if } v \notin \{u,\perp\} \text{ and } w \neq u \end{cases} \tag{4}$$

Note again that together with (2) this covers Case 1.2.

Case 2 ($i = r_u$ for some u) Now we have that $V_i = V_{i-1}$, and we need to distinguish whether $\mathsf{bd}(i)$ is a maximal clique or not.

Case 2.1 ($\mathsf{bd}(i)$ is not a maximal clique) In this case, $\mathcal{C}_i = \mathcal{C}_{i-1}$. Suppose first that $i-1 = \ell_x$ for some x, then we also have that $\mathsf{bd}(i) = \mathsf{bd}(i-1)$. Therefore, we simply let $c[i, v, w] = c[i-1, v, w]$. If $i-1 = r_x$ for some x, then $\mathsf{bd}(i) = \mathsf{bd}(i-1) \setminus \{x\}$ and we once more have to take into account that x may have been used by a solution when $v = \perp$. If $v \neq \perp$, then $v \in \mathsf{bd}(i-1)$. To wrap up this case, we have

$$c[i,v,w] = \begin{cases} \max\{c[i-1,x,w], c[i-1,\perp,w]\}, & \text{if } i-1 = r_x \text{ and } v = \perp \\ c[i-1,v,w], & \text{otherwise} \end{cases} \tag{5}$$

Case 2.2 ($\mathsf{bd}(i)$ is a maximal clique) In this case, since $i = r_u$, we have that $\mathcal{C}_i = \mathcal{C}_{i-1} \cup \{\mathsf{bd}(i)\}$. We can also observe that $i - 1 = \ell_x$ for some x, and therefore $\mathsf{bd}(i) = \mathsf{bd}(i-1)$. Indeed, if $i - 1 = r_x$ for some x, then $\mathsf{bd}(i-1) = \mathsf{bd}(i) \cup \{x\}$ would be a clique strictly containing $\mathsf{bd}(i)$, meaning that $\mathsf{bd}(i)$ was not maximal. Since $\mathsf{bd}(i)$ is a maximal clique, any solution has to have a vertex from $\mathsf{bd}(i)$; if a solution has only one vertex from $\mathsf{bd}(i)$, then this vertex must have $\mathsf{bd}(i)$ as its witness. Therefore, we let $c[i, \perp, w] = -\infty$ for any $w \in \mathsf{bd}(i) \cup \{\perp\}$. Next, suppose $v \neq \perp$ and consider $c[i, v, \perp]$. Then, v has $\mathsf{bd}(i)$ as its witness; however, v could have another witness strictly forgotten at i, so we let $c[i, v, \perp] = \max\{c[i-1, v, \perp], c[i-1, \perp, v]\}$. When $v \neq \perp$ and $w \neq \perp$, then any partial solution at i matching this index has two vertices from $\mathsf{bd}(i)$, therefore $\mathsf{bd}(i)$ cannot witness any vertex. This means that we simply let $c[i, v, w] = c[i-1, v, w]$. To summarize this case:

$$c[i,v,w] = \begin{cases} -\infty, & \text{if } v = \perp \\ \max\{c[i-1,v,\perp], c[i-1,\perp,v]\}, & \text{if } v \neq \perp \text{ and } w = \perp \\ c[i-1,v,w], & \text{if } v \neq \perp \text{ and } w \neq \perp \end{cases} \tag{6}$$

▶ **Lemma 14 (♣).** *Equation* (1) *and Algorithm 1 compute the table entries from Definition 13 correctly, meaning that $c[i, v, w] \geq k$ if and only if there is a partial solution of size k that matches the index (i, v, w).*

Using the linear-time algorithm due to Booth and Lueker [7] to construct a representation of the input graph and following the above DP algorithm we can prove the main result of this section. Correctness is due to Lemma 14.

▶ **Theorem 1.** *There is an algorithm that solves* Upper Clique Transversal *on n-vertex interval graphs in $\mathcal{O}(n^3)$ time.*

Algorithm 1 How to compute a table entry $c[i, v, w]$ for $i > 1$.

```
1  if i = ℓ_u for some u ∈ V(G) then
       % Case 1
2      if v = u then c[i, v, w] ← −∞;
3      else if v ∈ bd(i) and w = u then c[i, v, w] ← 1 + c[i − 1, v, ⊥];
4      else
5          if i − 1 = ℓ_x for some x then
               % Case 1.1
6              if v = ⊥ and w = u then c[i, v, w] ← 1 + c[i − 1, ⊥, ⊥];
7              else
                   % Note that v, w ∈ bd(i) \ {u} ∪ {⊥}.
8                  c[i, v, w] ← c[i − 1, v, w];
9          else
               % Case 1.2 (i = r_x for some x)
10             if v = ⊥ then
11                 if w = u then c[i, v, w] ← 1 + max{c[i − 1, x, ⊥], c[i − 1, ⊥, ⊥]};
12                 else c[i, v, w] ← max{c[i − 1, x, w], c[i − 1, ⊥, w]};
13             else
                   % Note that v ∈ bd(i) and therefore w ≠ u.
14                 c[i, v, w] ← c[i − 1, v, w];
15 else
       % Case 2 (i = r_u for some u)
16     if bd(i) is not a maximal clique then
           % Case 2.1
17         if i − 1 = r_x and v = ⊥ then c[i, v, w] ← max{c[i − 1, x, w], c[i − 1, ⊥, w]};
18         else c[i, v, w] ← c[i − 1, v, w];
19     else
           % Case 2.2 (bd(i) is a maximal clique)
20         if v = ⊥ then c[i, v, w] ← −∞;
21         else if w = ⊥ then c[i, v, w] ← max{c[i − 1, v, ⊥], c[i − 1, ⊥, v]};
22         else c[i, v, w] ← c[i − 1, v, w];
```

4 Rooted Directed Path Graphs

In this section, we give a polynomial-time algorithm for UCT on rooted directed path graphs. Rooted directed path graphs form a subclass of chordal graphs that generalize interval graphs. Recall that UCT is NP-complete on chordal graphs [21]. To simplify the description of the algorithm, we use the following special representation of the input graph.

▶ **Definition 15.** *Let G be a rooted directed path graph with representation $\mathcal{R} = (T, \{P_v\}_{v \in V(T)})$. For each $v \in V(G)$, we denote by $\ell_v \in V(T)$ the* lower *endpoint of P_v, and by $u_v \in V(T)$ the* upper *endpoint of P_v; i.e., ℓ_v is a descendant of u_v in T. For each $t \in V(T)$ we denote by $Q(t)$ the set of vertices whose path has an endpoint at t, meaning*

$$Q(t) = \{v \in V(G) \mid t \in \{\ell_v, u_v\}\}. \tag{7}$$

We call $\mathcal{R}$ canonical *if each node $t \in V(T)$ is one of the following:*

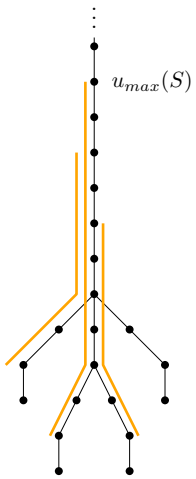

Figure 3 Illustration of Definition 18.

Buffer node *$Q(t) = \emptyset$ and t has at most one child; if it has a child, then this child is not a buffer node.*

Introduce node *$Q(t) = \{v\}$ and $t = \ell_v$; v is called the vertex* introduced *at t. Moreover, t has precisely one child and that child is a buffer node.*

Forget node *$Q(t) = \{v\}$ and $t = u_v$; v is called the vertex* forgotten *at t. Moreover, t has precisely one child and that child is a buffer node.*

Join node *$Q(t) = \emptyset$ and t has at least two children, none of which is an introduce or a forget node.*

Note that in a representation $(T, \{P_v\}_{v \in V(G)})$, since T is a tree, any path P_v is fully described by its two endpoints. Throughout the following, when manipulating the tree of a representation, we only describe the changes made to paths whose endpoints shift; modifications to paths whose endpoints do not change are not made explicit.

▶ **Lemma 16 (♣).** *Let G be a rooted directed path graph. Then G admits a canonical representation which can be obtained from any representation with tree T in $\mathcal{O}(|V(G)|+|V(T)|)$ time.*

Let G be a rooted directed path graph with canonical representation $\mathcal{R} = (T, \{P_v\}_{v \in V(G)})$. For $t \in V(T)$, we let T_t denote the subtree of T rooted at t, i.e., T_t is the subtree of T induced by t and all its descendants. In analogy with Definition 7 for interval graphs, we define $V_t = \{v \in V(G) \mid \ell_v \in V(T_t)\}$ and the boundary $\mathsf{bd}(t) = \{v \in V(G) \mid \ell_v \in V(T_t) \wedge u_v \in \{t\} \cup (V(T) \setminus V(T_t))\}$. For the blank symbol $\perp$, we use the shorthand $\mathsf{bd}_\perp(t)$ for $\mathsf{bd}(t) \cup \{\perp\}$.

Let $S \subseteq V(G)$ and let $t \in V(T)$. We say that S *is forgotten at t* if there is some $v \in S$ with $u_v \in V(T_t)$. For $t \in V(T)$, we denote by $\mathcal{C}_t$ the set of maximal cliques in G that are forgotten at t.

The algorithm we give is a generalization of the ideas from Section 3 to rooted directed path graphs. Once more, partial solutions at a node $t \in V(T)$ are hitting sets of $\mathcal{C}_t$, where everybody except up to one vertex has a witness. However, a direct analogue of Lemma 11 does not hold for rooted directed path graphs: For any node $t \in V(T)$, a minimal hitting set of $\mathcal{C}_t$ *may* contain distinct vertices v, w from $\mathsf{bd}(t)$, for instance if t is a join node and P_v and P_w go to different children of t. However, v and w have a comparable behavior with respect to the maximal cliques that will be forgotten in the future, as either the subpath of P_v from u_v to t is contained in the subpath of P_w from u_w to t or vice versa, cf. Figure 3.

▶ **Corollary 17** (cf. Corollary 12). *Let G be a rooted directed path graph with representation $\mathcal{R} = (T, \{P_v\}_{v \in V(G)})$. Let S be a minimal clique transversal of G and $t \in V(T)$. There is at most one vertex in $S \cap \mathsf{bd}(t)$ that does not have a witness forgotten at t.*

To formalize the table entries, we need the following definition, illustrated in Figure 3.

▶ **Definition 18.** *Let G be a rooted directed path graph with representation $\mathcal{R} = (T, \{P_v\}_{v \in V(G)})$. Let $S \subseteq V(G)$ be such that $\bigcap_{v \in S} V(P_v) \neq \emptyset$. Then, $u_{max}(S)$ is a vertex $v \in S$ such that u_v is the highest (closest to the root) node in T among $\{u_w \mid w \in S\}$. As a convention, we let $u_{max}(\emptyset) = \perp$.*[2]

Note in the previous definition that if $\mathcal{R}$ is canonical, then the choice for $u_{max}(S)$ is unique. We are now ready to define the table entries of our algorithm.[3]

▶ **Definition 19.** *Let G be a rooted directed path graph with canonical representation $\mathcal{R} = (T, \{P_v\}_{v \in V(G)})$. For $t \in V(T)$ and $v, w \in \mathsf{bd}(t) \cup \{\perp\}$, let $c[t, v, w]$ be the largest size of any set $S \subseteq V_t$ such that*

1. *S is a hitting set of $\mathcal{C}_t$,*
2. *$\{v, w\} \setminus \{\perp\} \subseteq S$,*
3. *$u_{max}((S \cap \mathsf{bd}(t)) \setminus \{w\}) = v$,*
4. *each vertex in $S \setminus \{w\}$ has a witness that is forgotten at t, and*
5. *if $w \neq \perp$, then w does not have a witness that is forgotten at t.*

We say that any set S satisfying the above five conditions is a partial solution that matches *(t, v, w). If no such S exists, then $c[t, v, w] = -\infty$.*

We now show how to compute the table entries bottom-up. First, consider the case when $t \in V(T)$ is a leaf of T. Since $\mathcal{R}$ is a canonical representation, we know that t is a buffer node; therefore, $V_t = \emptyset$, and the only table entry is $c[t, \perp, \perp] = 0$.

From now on, we may assume that $t \in V(T)$ has at least one child and we let $v, w \in \mathsf{bd}(t) \cup \{\perp\}$. We distinguish the following cases, based on the type of node that t is.

Introduce Let x be the vertex introduced at t, and let t' denote the child of t in T. We have that $V_t = V_{t'} \cup \{x\}$, $\mathcal{C}_t = \mathcal{C}_{t'}$, and since t' is a buffer node, $\mathsf{bd}(t) = \mathsf{bd}(t') \cup \{x\}$. Since x is not part of any forgotten maximal clique, x cannot have any witness forgotten at t. Therefore, if $v = x$, we have that $c[t, v, w] = -\infty$. Second, if $w = x$, then $c[t, v, w] = 1 + c[t', v, \perp]$; as x can be included to any partial solution at t' where all vertices have witnesses forgotten at t'. (But not to any partial solution at t' that already had a vertex without a witness, see Corollary 17.) If $v \neq x$ and $w \neq x$, since $\mathsf{bd}(t) \setminus \{x\} = \mathsf{bd}(t')$, we can look up the solution at the previous table.

$$c[t, v, w] = \begin{cases} -\infty, & \text{if } v = x \\ 1 + c[t', v, \perp], & \text{if } w = x \\ c[t', v, w], & \text{otherwise} \end{cases} \tag{8}$$

Forget Let x be the vertex forgotten at t, and let t' be the child of t in T. In this case, $V_t = V_{t'}$ and $\mathsf{bd}(t) = \mathsf{bd}(t')$. Moreover, $\mathsf{bd}(t)$ is a clique that is forgotten at t. If $\mathsf{bd}(t)$ is not maximal, then once more $\mathcal{C}_t = \mathcal{C}_{t'}$ and we can simply let $c[t, v, w] = c[t', v, w]$. If

[2] Recall that we use $\perp$ as a "blank" symbol and that we may use the convention $\{\perp\} = \emptyset$.

[3] Observe that the algorithm for interval graphs can be formulated in the same way as well, but we find the formulation in Section 3 more straightforward.

$\mathsf{bd}(t)$ is maximal, then $\mathcal{C}_t = \mathcal{C}_{t'} \cup \{\mathsf{bd}(t)\}$ and any partial solution must include a vertex from $\mathsf{bd}(t)$ (that hence has a witness forgotten at t). Therefore, we set $c[t, \bot, w] = -\infty$. Moreover, if $v \neq \bot$ and $w = \bot$, then $\mathsf{bd}(t)$ can serve as a witness for v. We set $c[t, v, \bot] = \max\{c[t', v, \bot], c[t', \bot, v]\}$. (Note that v may already have had another witness previously.) The remaining cases can be copied from the previous table. To summarize, in the case when $\mathsf{bd}(t)$ is not maximal, we set $c[t, v, w] = c[t', v, w]$, and if $\mathsf{bd}(t)$ is maximal, we set:

$$c[t, v, w] = \begin{cases} -\infty, & \text{if } v = \bot, \\ \max\{c[t', v, \bot],\, c[t', \bot, v]\}, & \text{if } w = \bot, \\ c[t', v, w], & \text{otherwise.} \end{cases} \tag{9}$$

Buffer Suppose t is a buffer node with child t'. Since no vertex is introduced or forgotten at t, $V_t = V_{t'}$ and $\mathcal{C}_t = \mathcal{C}_{t'}$. We distinguish the following cases. If t' is an introduce or a join node, then also $\mathsf{bd}(t) = \mathsf{bd}(t')$, so we can simply consult the table at t'. Suppose t' is a forget node, and let x be the vertex forgotten at t'. Then, $\mathsf{bd}(t) = \mathsf{bd}(t') \setminus \{x\}$, and if $v = \bot$, then any partial solution matching (t, v, w) may or may not include x. So we set $c[t, \bot, w] = \max\{c[t', \bot, w], c[t', x, w]\}$. For all other pairs (v, w), we may again copy the entry from the previous table. In summary:

$$c[t, v, w] = \begin{cases} \max\{c[t', \bot, w], c[t', x, w]\}, & \text{if } t' \text{ is a forget node forg. } x \text{ and } v = \bot \\ c[t', v, w], & \text{otherwise} \end{cases} \tag{10}$$

The last case to consider is when t is a join node. Before we proceed with the description of this case, we make a simple but crucial observation which follows from the fact that no vertex is introduced or forgotten at a join node.

▶ **Observation 20.** *Let $t \in V(T)$ be a join node with children $t_1, \ldots, t_d$, $d \geq 2$.*

1. *$\mathcal{C}_t = \mathcal{C}_{t_1} \dot\cup \cdots \dot\cup \mathcal{C}_{t_d}$.*
2. *For $i \in [d]$, let S_i be a hitting set for $\mathcal{C}_{t_i}$. Then, $S = \bigcup_{i \in [d]} S_i$ is a hitting set for $\mathcal{C}_t$. Moreover, a vertex $v \in S$ has a witness forgotten at t if and only if it has a witness forgotten at t_i, where $v \in V_{t_i}$.*
3. *Let S be a hitting set for $\mathcal{C}_t$. Then, for all $i \in [d]$, $S_i = S \cap V_{t_i}$ is a hitting set for $\mathcal{C}_{t_i}$. Moreover, a vertex $v \in S_i$ has a witness forgotten at t_i if and only if it has a witness forgotten at t.*

Join Suppose t is a join node with children $t_1, \ldots, t_d$, $d \geq 2$. Since t is a join node, no vertex is introduced or forgotten at t or any of its children. We therefore have that $V_t = \dot\bigcup_{i \in [d]} V_{t_i}$ and $\mathsf{bd}(t) = \dot\bigcup_{i \in [d]} \mathsf{bd}(t_i)$. By Observation 20, we know that all partial solutions at t are obtained as disjoint unions of partial solutions at $t_1, \ldots, t_d$. In such a union, there may be precisely one child t_i such that the partial solution at t_i has a vertex without a witness (if $w \neq \bot$), see Corollary 17. If $w \neq \bot$, let t_w be the child of t such that $\ell_w \in V(T_{t_w})$; we let $t_w = \bot$ otherwise. Furthermore, if $v \neq \bot$, we let t_v be the child of t such that $\ell_v \in V(T_{t_v})$; and we let $t_v = \bot$ otherwise. Note that Observation 20 tells us that for all children $t' \neq t_v$, we can independently choose the largest partial solution $S_{t'}$ at t' such that $u_{max}(S_{t'}) < v$.[4] In $S_{t'}$, all vertices need to have witnesses though.

[4] We write $w < v$ if u_w is a descendant of u_v in T. As a convention, we let $\bot < \bot$ and $w \not< \bot$ for all $w \neq \bot$.

Note that we have to distinguish the cases when $t_v = t_w$ and when $t_v \neq t_w$ to address the tables correctly.So, if $t_v = t_w$, we set $c[t, v, w]$ to

$$c[t_v, v, w] + \sum\nolimits_{t' \in \mathsf{ch}(t) \setminus \{t_v\}} \max\nolimits_{v' \in \mathsf{bd}_\perp(t'), v' < v} c[t', v', \perp], \tag{11}$$

and if $t_v \neq t_w$, we set it to

$$\begin{aligned} c[t_v, v, \perp] + \max\nolimits_{v' \in \mathsf{bd}_\perp(t_w), v' < v} c[t_w, v', w] \\ + \sum\nolimits_{t' \in \mathsf{ch}(t) \setminus \{t_v, t_w\}} \max\nolimits_{v' \in \mathsf{bd}_\perp(t'), v' < v} c[t', v', \perp]. \end{aligned} \tag{12}$$

To observe that the above algorithm is correct, notice that the ideas to compute the introduce, forget, and buffer nodes are nearly identical to the ones used in the algorithm for interval graphs. The correctness of the join node essentially follows from Observation 20.

▶ **Lemma 21 (♣).** *Equations* (8)–(12) *compute the table entries shown in Definition 19 in their respective cases correctly.*

Using the algorithm of Gavril [17] and Lemma 16 to obtain a suitable representation of the input graph, and then following the above DP algorithm which is correct by Lemma 21, we get the main result of this section.

▶ **Theorem 2.** *There is an algorithm that solves* Upper Clique Transversal *on n-vertex rooted directed path graphs in $\mathcal{O}(n^5)$ time.*

5 Conclusion

In this work, we gave a polynomial-time algorithm for UCT on interval graphs. A class of graphs that is often mentioned in the same breath as interval graphs is permutation graphs; on this class, the complexity of UCT is still open. A common generalization of interval and permutation graphs is the class of graphs with *linear mim-width* 1 [6], and rooted directed path graphs have (non-linear) mim-width 1. A nice extension of several of the main results in this work would therefore be a polynomial-time algorithm on graphs of mim-width 1.

▶ **Open Question 1.** *Is* Upper Clique Transversal *polynomial-time solvable on permutation graphs or, more generally, on graphs of (linear) mim-width* 1*?*

It would also be interesting to investigate the complexity of UCT on other generalizations of rooted directed path graphs, such as directed path graphs, leaf powers, or *strongly chordal* graphs. We observe that the graphs in the NP-hardness proof for chordal graphs due to Milanič and Uno [21] (cf. Theorem 3) are *not* strongly chordal, as they may contain k-suns[5] which are known obstructions for strongly chordal graphs [16].

We believe that our algorithm for interval graphs can be generalized to show that UCT is polynomial-time solvable on graphs of constant simultaneous interval number [5], given a representation of the input graph. A straightforward generalization would lead to an XP-algorithm in the language of parameterized complexity [11]; it would also be interesting to see if UCT is W[1]-hard in this parameterization, or if it could potentially have an FPT-algorithm.

▶ **Open Question 2.** *Is* Upper Clique Transversal *parameterized by the simultaneous interval number, given a representation of the input graph, in* XP *and* W[1]*-hard?*

[5] For $k \geq 3$, a *k-sun* is a split graph with $2k$ vertices, partitioned into two subsets $X = \{x_0, \ldots, x_{k-1}\}$, $Y = \{y_0, \ldots, y_{k-1}\}$ such that X is a clique, Y is independent, and each vertex $y_i \in Y$ has exactly two neighbors x_i, $x_{(i+1) \bmod k}$ in X.

References

1 Hassan AbouEisha, Shahid Hussain, Vadim Lozin, Jérôme Monnot, Bernard Ries, and Viktor Zamaraev. Upper domination: Towards a dichotomy through boundary properties. *Algorithmica*, 80:2799–2817, 2018. doi:10.1007/S00453-017-0346-9.

2 Jungho Ahn, Lars Jaffke, O-joung Kwon, and Paloma T. Lima. Well-partitioned chordal graphs. *Discrete Mathematics*, 345:112985, 2022. doi:10.1016/j.disc.2022.112985.

3 Thomas Andreae and Carsten Flotow. On covering all cliques of a chordal graph. *Discrete Mathematics*, 149(1-3):299–302, 1996. doi:10.1016/0012-365X(94)00276-O.

4 Cristina Bazgan, Ljiljana Brankovic, Katrin Casel, Henning Fernau, Klaus Jansen, Kim-Manuel Klein, Michael Lampis, Mathieu Liedloff, Jérôme Monnot, and Vangelis Th Paschos. The many facets of upper domination. *Theoretical Computer Science*, 717:2–25, 2018. doi:10.1016/J.TCS.2017.05.042.

5 Jesse Beisegel, Nina Chiarelli, Ekkehard Köhler, Martin Milanič, Peter Muršič, and Robert Scheffler. The simultaneous interval number: A new width parameter that measures the similarity to interval graphs. In Hans L. Bodlaender, editor, *Proceedings of the 19th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2024*, volume 294 of *LIPIcs*, pages 7:1–7:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.SWAT.2024.7.

6 Rémy Belmonte and Martin Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theoretical Computer Science*, 511:54–65, 2013. doi:10.1016/J.TCS.2013.01.011.

7 Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.

8 Nicolas Boria, Federico Della Croce, and Vangelis Th Paschos. On the max min vertex cover problem. *Discrete Applied Mathematics*, 196:62–71, 2015. doi:10.1016/J.DAM.2014.06.001.

9 Gerard J Chang, Martin Farber, and Zsolt Tuza. Algorithmic aspects of neighborhood numbers. *SIAM Journal on Discrete Mathematics*, 6(1):24–29, 1993. doi:10.1137/0406002.

10 Jou-Ming Chang. Induced matchings in asteroidal triple-free graphs. *Discrete Applied Mathematics*, 132(1):67–78, 2003. doi:10.1016/S0166-218X(03)00390-1.

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

12 Clément Dallard, Martin Milanič, and Kenny Štorgel. Treewidth versus clique number. II. Tree-independence number. *Journal of Combinatorial Theory, Series B*, 164:404–442, 2024. doi:10.1016/j.jctb.2023.10.006.

13 Peter Damaschke. Parameterized algorithms for double hypergraph dualization with rank limitation and maximum minimal vertex cover. *Discrete Optimization*, 8(1):18–24, 2011. doi:10.1016/J.DISOPT.2010.02.006.

14 Louis Dublois, Tesshu Hanaka, Mehdi Khosravian Ghadikolaei, Michael Lampis, and Nikolaos Melissinos. (In) approximability of maximum minimal FVS. *Journal of Computer and System Sciences*, 124:26–40, 2022. doi:10.1016/J.JCSS.2021.09.001.

15 Paul Erdős, Tibor Gallai, and Zsolt Tuza. Covering the cliques of a graph with vertices. *Discrete Mathematics*, 108(1-3):279–289, 1992. doi:10.1016/0012-365X(92)90681-5.

16 Martin Farber. Characterizations of strongly chordal graphs. *Discrete Mathematics*, 43(2-3):173–189, 1983. doi:10.1016/0012-365X(83)90154-1.

17 Fănică Gavril. A recognition algorithm for the intersection graphs of directed paths in directed trees. *Discrete Mathematics*, 13(3):237–249, 1975. doi:10.1016/0012-365X(75)90021-7.

18 Michael S Jacobson and Ken Peters. Chordal graphs and upper irredundance, upper domination and independence. *Discrete Mathematics*, 86(1-3):59–69, 1990. doi:10.1016/0012-365X(90)90349-M.

19 Kaveh Khoshkhah, Mehdi Khosravian Ghadikolaei, Jérôme Monnot, and Florian Sikora. Weighted upper edge cover: Complexity and approximability. *Journal of Graph Algorithms and Applications*, 24(2):65–88, 2020. doi:10.7155/JGAA.00519.

20 Michael Lampis, Nikolaos Melissinos, and Manolis Vasilakis. Parameterized Max Min Feedback Vertex Set. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*, volume 272 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 62:1–62:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2023.62.

21 Martin Milanič and Yushi Uno. Upper clique transversals in graphs. In Daniël Paulusma and Bernard Ries, editors, *Proceedings of the 49th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2023*, volume 14093 of *Lecture Notes in Computer Science*, pages 432–446. Springer, 2023. doi:10.1007/978-3-031-43380-1_31.

22 Clyde L Monma and Victor K Wei. Intersection graphs of paths in a tree. *Journal of Combinatorial Theory, Series B*, 41(2):141–181, 1986. doi:10.1016/0095-8956(86)90042-0.

23 Jérôme Monnot, Henning Fernau, and David Manlove. Algorithmic aspects of upper edge domination. *Theoretical Computer Science*, 877:46–57, 2021. doi:10.1016/J.TCS.2021.03.038.

24 Meirav Zehavi. Maximum minimal vertex cover parameterized by vertex cover. *SIAM Journal on Discrete Mathematics*, 31(4):2440–2456, 2017. doi:10.1137/16M109017X.