Polynomial-Time Algorithms for the Longest Induced Path and Induced Disjoint Paths Problems on Graphs of Bounded Mim-Width*†

Lars Jaffke‡[1], O-joung Kwon§[2], and Jan Arne Telle[3]

1 **Department of Informatics, University of Bergen, Norway**
`lars.jaffke@uib.no`
2 **Logic and Semantics, Technische Universität Berlin, Berlin, Germany**
`ojoungkwon@gmail.com`
3 **Department of Informatics, University of Bergen, Norway**
`jan.arne.telle@uib.no`

Abstract

We give the first polynomial-time algorithms on graphs of bounded *maximum induced matching width* (mim-width) for problems that are not locally checkable. In particular, we give $n^{\mathcal{O}(w)}$-time algorithms on graphs of mim-width at most w, when given a decomposition, for the following problems: Longest Induced Path, Induced Disjoint Paths and H-Induced Topological Minor for fixed H. Our results imply that the following graph classes have polynomial-time algorithms for these three problems: Interval and Bi-Interval graphs, Circular Arc, Permutation and Circular Permutation graphs, Convex graphs, k-Trapezoid, Circular k-Trapezoid, k-Polygon, Dilworth-k and Co-k-Degenerate graphs for fixed k.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, Computations on discrete structures, G.2.2 Graph Theory, Graph algorithms

Keywords and phrases graph width parameters, dynamic programming, graph classes, induced paths, induced topological minors

Digital Object Identifier 10.4230/LIPIcs.IPEC.2017.21

1 Introduction

Ever since the definition of the tree-width of graphs emerged from the Graph Minors project of Robertson and Seymour, bounded-width structural graph decompositions have been a successful tool in designing fast algorithms for graph classes on which the corresponding width-measure is small. Over the past few decades, many more width-measures have been introduced, see e.g. [8] for an excellent survey and motivation for width-parameters of graphs. In 2012, Vatshelle [18] defined the *maximum induced matching width* (mim-width for short) which measures how easy it is to decompose a graph along vertex cuts with bounded maximum induced matching size on the bipartite graph induced by edges crossing the cut. One interesting aspect of this width-measure is that its modeling power is much stronger than

* The work was done while the authors were at Polytechnic University of Valencia, Spain.
† A full version of the paper is available at `https://arxiv.org/abs/1708.04536`.
‡ Lars Jaffke is supported by the Bergen Research Foundation (BFS).
§ O-joung Kwon is Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC consolidator grant DISTRUCT, agreement No. 648527).

tree-width and clique-width and many well-known and deeply studied graph classes such as Interval graphs and Permutation graphs have (linear) mim-width 1, with decompositions that can be found in polynomial time [2, 18], while their clique-width can be proportional to the square root of the number of vertices. Hence, designing an algorithm for a problem Π that runs in XP time parameterized by mim-width yields polynomial-time algorithms for Π on several interesting graph classes at once.

For Locally Checkable Vertex Subset and Vertex Partitioning (LC-VSVP) problems, a class introduced [17] to capture many well-studied algorithmic problems in a unified framework, Belmonte and Vatshelle [2] and Bui-Xuan et al. [3] provided XP-algorithms on graphs of bounded mim-width. LC-VSVP problems include many NP-hard problems such as Maximum Independent Set, Minimum Dominating Set, and q-Coloring. A common feature of these problems is that they (as the name suggests) can be checked locally: Take q-Coloring for example. Here, we want to determine whether there is a q-partition of the vertex set of an input graph such that each part induces an independent set. The latter property can be checked individually for each vertex by inspecting only its direct neighborhood.

Until now, the only problems known to be XP-time solvable on graphs of bounded mim-width were of the type LC-VSVP. It is therefore natural to ask whether similar results can be shown for problems concerning graph properties that are *not* locally checkable. In this paper, we study problems related to finding *induced* paths in graphs, namely Longest Induced Path, Induced Disjoint Paths and H-Induced Topological Minor. Although their 'non-induced' counterparts are more deeply studied in the literature, also these induced variants have received considerable attention. Below, we briefly survey results for exact algorithms to these three problems on graph classes studied so far.

For the first problem, Gavril [5] showed that Longest Induced Path can be solved in polynomial time for graphs without induced cycles of length at least q for fixed q (the running time was improved by Ishizeki et al. [9]), while Kratsch et al. [13] solved the problem on AT-free graphs in polynomial time. Kang et al. [11] recently showed that those classes have unbounded mim-width. However, graphs of bounded mim-width are not necessarily graphs without cycles of length at least k or AT-free graphs. The second problem derives from the well-known Disjoint Paths problem which is solvable in $O(n^3)$ time if the number of paths k is a fixed constant, as shown by Robertson and Seymour [15], while if k is part of the input it is NP-complete on graphs of linear mim-width 1 (interval graphs) [14]. In contrast, Induced Disjoint Paths is NP-complete already for $k = 2$ paths [12]. In this paper we consider the number of paths k as part of the input. Under this restriction Induced Disjoint Paths is NP-complete on claw-free graphs, as shown by Fiala et al. [4], while Golovach et al. [7] gave a linear-time algorithm for circular-arc graphs. For the third problem, H-Induced Topological Minor, we consider H to be a fixed graph. This problem, and also Induced Disjoint Paths, were both shown solvable in polynomial time on chordal graphs by Belmonte et al. [1], and on AT-free graphs by Golovach et al. [6].

We show that Longest Induced Path, Induced Disjoint Paths and H-Induced Topological Minor for fixed H can be solved in time $n^{\mathcal{O}(w)}$ given a branch decomposition of mim-width w. Since bounded mim-width decompositions, usually mim-width 1 or 2, can be computed in polynomial-time for all well-known graph classes having bounded mim-width [2], our results thus provide unified polynomial-time algorithms for these problems on the following classes of graphs: Interval and Bi-Interval graphs, Circular Arc, Permutation and Circular Permutation graphs, Convex graphs, k-Trapezoid, Circular k-Trapezoid, k-Polygon, Dilworth-k and Co-k-Degenerate graphs for fixed k, all graph classes of bounded mim-width [2].

The problem of computing the mim-width of general graphs was shown to be W[1]-hard [16] and no algorithm for computing the mim-width of a graph in XP time is known. Furthermore, there is no polynomial-time constant-factor approximation for mim-width unless NP = ZPP [16].

What makes our algorithms work is an analysis of the structure induced by a solution to the problem on a cut in the branch decomposition. There are two ingredients. First, in all the problems we investigate, we are able to show that for each cut induced by an edge of the given branch decomposition of an input graph, it is sufficient to consider induced subgraphs of size at most $\mathcal{O}(w)$ as intersections of solutions and the set of edges crossing the cut, where w is the mim-width of the branch decomposition. For instance, in the Longest Induced Paths problem, an induced path is a target solution. We argue that an induced path cannot cross a cut many times if there is no large induced matching between vertex sets A and B of the cut (A, B). Such an intersection is always a disjoint union of paths. Thus, we enumerate all subgraphs of size at most $\mathcal{O}(w)$, which are disjoint unions of paths, and these will be used as indices of our table.

However, a difficulty arises if we recursively ask for a given cut and such an intersection of size at most $\mathcal{O}(w)$, whether there is an induced disjoint union of paths of certain size in the union of one part and edges crossing the cut, whose intersection on the crossing edges is the given subgraph. The reason is that there are unbounded number of vertices in each part of the cut that are not contained in the given subgraph of size $\mathcal{O}(w)$ but still have neighbors in the other part. We need to control these vertices in such a way that they do not further create an edge in the solution. We control these vertices using vertex covers of the bipartite graph induced by edges crossing the cut. Roughly speaking, if there is a valid partial solution, then there is a vertex cover of such a bipartite graph, which meets all other edges not contained in the given subgraph. The point is that there are only $n^{\mathcal{O}(w)}$ many minimal vertex covers of such a bipartite graph with maximum matching size w. We discuss this property in Section 2. Based on these two results, each table will consist of a subgraph of size $\mathcal{O}(w)$ and a vertex cover of the remaining part of the bipartite graph, and we check whether there is a (valid) partial solution to the problem with respect to given information. We can argue that we need to store at most $n^{\mathcal{O}(w)}$ table entries in the resulting dynamic programming scheme and that each of them can be computed in time $n^{\mathcal{O}(w)}$ as well.

The strategy for Induced Disjoint Paths is very similar to the one for Longest Induced Path. The only thing to additionally consider is that in the disjoint union of paths, which is guessed as the intersection of a partial solution and edges crossing a cut, we need to remember which path is a subpath of the path connecting a given pair. We lastly provide a one-to-many reduction from H-Induced Topological Minor to Induced Disjoint Paths, that runs in polynomial time, and show that it can be solved in time $n^{\mathcal{O}(w)}$. Similar reductions have been shown earlier (see e.g. [1, 6]) but we include it here for completeness.

Throughout the paper, proofs of statements marked with '★' are deferred to the full version [10].

2 Preliminaries

For integers a and b with $a \leq b$, we let $[a..b] := \{a, a+1, \ldots, b\}$ and if a is positive, we define $[a] := [1..a]$. Throughout the paper, a graph G on vertices $V(G)$ and edges $E(G) \subseteq \binom{V(G)}{2}$ is assumed to be finite, undirected and simple. For graphs G and H we say that G is a *subgraph* of H, if $V(G) \subseteq V(H)$ and $E(G) \subseteq E(H)$ and we write $G \subseteq H$. For a vertex set $X \subseteq V(G)$, we denote by $G[X]$ the subgraph *induced* by X, i.e. $G[X] := (X, E(G) \cap \binom{X}{2})$. If

$H \subseteq G$ and $X \subseteq V(G)$ then we let $H[X] := H[X \cap V(H)]$. We use the shorthand $G - X$ for $G[V(G) \setminus X]$. For two (disjoint) vertex sets $X, Y \subseteq V(G)$, we denote by $G[X, Y]$ the bipartite subgraph of G with bipartition (X, Y) such that for $x \in X, y \in Y$, x and y are adjacent in G if and only if they are adjacent in $G[X, Y]$. A *cut* of G is a bipartition (A, B) of its vertex set. For a vertex $v \in V(G)$, we denote by $N[v]$ the set of *neighbors* of v in G, i.e. $N[v] := \{w \in V(G) \mid \{v, w\} \in E(G)\}$. A set M of edges is a *matching* if no two edges in M share an end vertex, and a matching $\{a_1b_1, \ldots, a_kb_k\}$ is *induced* if there are no other edges in the subgraph induced by $\{a_1, b_1, \ldots, a_k, b_k\}$. For an edge $e = \{v, w\} \in E(G)$, the operation of *contracting* e is to remove the edge e from G and merging its endpoints v and w.

Branch Decompositions and Mim-Width. A pair $(T, \mathcal{L})$ of a subcubic tree T and a bijection $\mathcal{L}$ from $V(G)$ to the set of leaves of T is called a *branch decomposition*. For each edge e of T, let T_1^e and T_2^e be the two connected components of $T - e$, and let (A_1^e, A_2^e) be the vertex bipartition of G such that for each $i \in \{1, 2\}$, A_i^e is the set of all vertices in G mapped to leaves contained in T_i^e by $\mathcal{L}$. The *mim-width of* $(T, \mathcal{L})$, denoted by $\mathrm{mimw}(T, \mathcal{L})$, is defined as $\max_{e \in E(T)} \mathrm{mim}(A_1^e)$, where for a vertex set $A \subseteq V(G)$, $\mathrm{mim}(A)$ denotes the maximum size of an induced matching in $G[A, V(G) \setminus A]$. The minimum mim-width over all branch decompositions of G is called the *mim-width of* G. If $|V(G)| \leq 1$, then G does not admit a branch decomposition, and the mim-width of G is defined to be 0.

To avoid confusion, we refer to elements in $V(T)$ as *nodes* and elements in $V(G)$ as *vertices* throughout the rest of the paper. Given a branch decomposition, one can subdivide an arbitrary edge and let the newly created vertex be the root of T, in the following denoted by r. Throughout the following we assume that each branch decomposition has a root node of degree two. For two nodes $t, t' \in V(T)$, we say that t' is a *descendant* of t if t lies on the path from r to t' in T. For $t \in V(T)$, we denote by G_t the subgraph induced by all vertices that are mapped to a leaf that is a descendant of t, i.e. $G_t = G[X_t]$, where $X_t = \{v \in V(G) \mid \mathcal{L}^{-1}(t') = v \text{ where } t' \text{ is a descendant of } t \text{ in } T\}$. We use the shorthand '$V_t$' for '$V(G_t)$' and let $\bar{V}_t := V(G) \setminus V_t$.

The following definitions which we relate to branch decompositions of graphs will play a central role in the design of the algorithms in Section 3.

▶ **Definition 1** (Boundary). Let G be a graph and $A, B \subseteq V(G)$ such that $A \cap B = \emptyset$. We let $\mathrm{bd}_B(A)$ be the set of vertices in A that have a neighbor in B, i.e. $\mathrm{bd}_B(A) := \{v \in V(A) \mid N(v) \cap B \neq \emptyset\}$. We define $\mathrm{bd}(A) := \mathrm{bd}_{V(G) \setminus A}(A)$ and call $\mathrm{bd}(A)$ the *boundary* of A in G.

▶ **Definition 2** (Crossing Graph). Let G be a graph and $A, B \subseteq V(G)$. If $A \cap B = \emptyset$, we define the graph $G_{A,B} := G[\mathrm{bd}_B(A), \mathrm{bd}_A(B)]$ to be the *crossing graph from A to B*.

If $(T, \mathcal{L})$ is a branch decomposition of G and $t_1, t_2 \in V(T)$ such that the crossing graph $G_{V_{t_1}, V_{t_2}}$ is defined, we use the shorthand $G_{t_1,t_2} := G_{V_{t_1}, V_{t_2}}$. We use the analogous shorthand notations $G_{t_1,\bar{t_2}} := G_{V_{t_1}, \bar{V_{t_2}}}$ and $G_{\bar{t_1},t_2} := G_{\bar{V_{t_1}}, V_{t_2}}$ (whenever these graphs are defined). For the frequently arising case when we consider $G_{t,\bar{t}}$ for some $t \in V(T)$, we refer to this graph as the *crossing graph w.r.t. t*.

We furthermore use the following notation. Let G be a graph, $v \in V(G)$ and $A \subseteq V(G)$. We denote by $N_A[v]$ the set of neighbors of v in A, i.e. $N_A[v] := N[v] \cap A$. For $X \subseteq V(G)$, we let $N_A[X] := \bigcup_{v \in X} N_A[v]$. If $(T, \mathcal{L})$ is a branch decomposition of G and $t \in V(T)$, we use the shorthand notations $N_t[X] := N_{V_t}[X]$ and $N_{\bar{t}}[X] := N_{\bar{V_t}}[X]$.

The Minimal Vertex Covers Lemma. Let G be a graph. We now prove that given a set $A \subseteq V(G)$, the number of minimal vertex covers in $G[A, V(G) \setminus A]$ is bounded by $n^{\mathrm{mim}(A)}$.

This observation is crucial to argue that we only need to store $n^{\mathcal{O}(w)}$ entries at each node in the branch decomposition in all algorithms we design, where w is the mim-width of the given branch decomposition.

▶ **Corollary 3** (Minimal Vertex Covers Lemma, ★). *Let H be a bipartite graph on n vertices with a bipartition (A, B). The number of minimal vertex covers of H is at most $n^{\mathrm{mim}(A)}$, and the set of all minimal vertex covers of H can be enumerated in time $n^{\mathcal{O}(\mathrm{mim}(A))}$.*

3 Algorithms

In all algorithms presented in this section, we assume that we are given as input an undirected graph G together with a branch decomposition $(T, \mathcal{L})$ of G of mim-width w, rooted at a degree two vertex obtained from subdividing an arbitrary edge in T. We do bottom-up dynamic programming over $(T, \mathcal{L})$. To obtain our algorithms, we study the structure a solution induces across a cut in the branch decomposition and argue that the size of this structure is bounded by a function only depending on the mim-width. The table entries at each node $t \in V(T)$ are then indexed by all possible such structures and contain the value 1 if and only if the structure used as the index of this entry constitutes a solution for the respective problem. After applying the dynamic programming scheme, the solution to the problem can be obtained by inspecting the table values associated with the root of T.

The rest of this section is organized as follows. In Section 3.1 we present an $n^{\mathcal{O}(w)}$-time algorithm for Longest Induced Path, and in Section 3.2 we give an algorithm for Induced Disjoint Paths with the same asymptotic runtime bound. We give a polynomial-time one-to-many reduction from H-Induced Topological Minor (for fixed H) to Induced Disjoint Paths in Section 3.3, yielding an $n^{\mathcal{O}(w)}$ for the former problem as well.

3.1 Longest Induced Path

For a disjoint union of paths P, we refer to its *size* as the number of its vertices, i.e. $|P| := |V(P)|$. If P has only one component, we use the terms 'size' and 'length' interchangeably. We now give an $n^{\mathcal{O}(w)}$ time algorithm for the following parameterized problem.

Longest Induced Path (LIP)/Mim-Width
Input: A graph G with branch decomposition $(T, \mathcal{L})$ and an integer k
Parameter: $w := \mathrm{mimw}(T, \mathcal{L})$
Question: Does G contain an induced path of length at least k?

Before we describe the algorithm, we observe the following. Let G be a graph and $A \subseteq V(G)$ with $\mathrm{mim}(A) = w$ and let P be an induced path in G. Then the subgraph induced by edges of P in $G_{A,\bar{A}}$ and vertices incident with these edges has size linearly bounded by w. The following lemma provides a bound of this size.

▶ **Lemma 4.** *Let p be a positive integer and let F be a disjoint union of paths such that each component of F contains an edge. If $|V(F)| \geq 4p$, then F contains an induced matching of size at least p.*

Proof. We prove the lemma by induction on p. If $p = 1$, then it is clear. We may assume $p \geq 2$. Suppose F contains a connected component C with at most 4 vertices. Then $F - V(C)$ contains at least $4(p-1)$ vertices, and thus it contains an induced matching of size at least $p-1$ by the induction hypothesis. As C contains an edge, F contains an induced matching of size at least p. Thus, we may assume that each component of F contains at least 5 vertices.

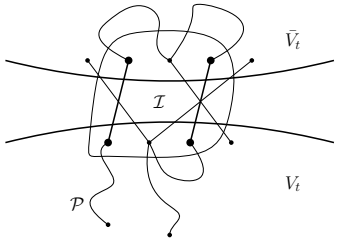

Figure 1 The intersection of an induced path $\mathcal{P}$ with $G[V_t \cup \mathrm{bd}(\bar{V}_t)]$, which is an induced disjoint union of paths $\mathcal{I}$. The subgraph S to be used as an index for the corresponding table entry consists of the boldface vertices and edges in $\mathcal{I}$.

Let us choose a leaf v of F, and let v_1 be the neighbor of v, and v_2 be the neighbor of v_1 other than v. Since each component of $F - \{v, v_1, v_2\}$ contains at least one edge, we can apply induction to conclude that $F - \{v, v_1, v_2\}$ contains an induced matching of size at least $p - 1$. Together with vv_1, F contains an induced matching of size at least p. ◀

Before we give the description of the dynamic programming algorithm, we first observe how a solution $\mathcal{P}$, i.e. an induced path in G, interacts with the graph $G[V_t \cup \mathrm{bd}(\bar{V}_t)]$, for some $t \in V(T)$. The intersection of $\mathcal{P}$ with $G[V_t \cup \mathrm{bd}(\bar{V}_t)]$ is an induced disjoint union of paths which we will denote by $\mathcal{I}$ in the following. To keep the number of possible table entries bounded by $n^{\mathcal{O}(w)}$, we have to focus on the interaction of $\mathcal{I}$ with the crossing graph $G_{t,\bar{t}}$ w.r.t. t, in particular the intersection of $\mathcal{I}$ with its edges. Note that after removing isolated vertices, $\mathcal{I}$ induces a disjoint union of paths on $G_{t,\bar{t}}$ which throughout the following we will denote by S. For an illustration see Figure 1. There cannot be any additional edges crossing the cut $(V_t, \bar{V}_t)$ between vertices in $\mathcal{I}$ on opposite sides of the boundary that are not contained in $V(S)$. This property of $\mathcal{I}$ can be captured by considering a minimal vertex cover M of the bipartite graph $G_{t,\bar{t}} - V(S)$. We remark that the vertices in M play different roles, depending on whether they lie in $M \cap V_t$ or $M \cap \bar{V}_t$. We therefore define the following two sets.

- $M_t^{\mathrm{in}} := M \cap V_t$ is the set of vertices that must be avoided by $\mathcal{I}$.
- $M_t^{\mathrm{out}} := M \cap \bar{V}_t$ is the set of vertices that must be avoided by a partial solution (e.g. the intersection of $\mathcal{P}$ with $G[\bar{V}_t]$) to be combined with $\mathcal{I}$ to ensure that their combination does not use any edges in $G_{t,\bar{t}} - V(S)$.

Furthermore, $\mathcal{I}$ also indicates how the vertices in $S[V_t]$ that have degree one in S are joined together in the graph G_t (possibly outside $\mathrm{bd}(V_t)$). This gives rise to a collection of vertex pairs Q, which we will refer to as *pairings*, with the interpretation that $(s, t) \in Q$ if and only if there is a path from s to t in $\mathcal{I}[V_t]$.

The description given above immediately tells us how to index the table entries in the dynamic programming table $\mathcal{T}$ to keep track of all possible partial solutions in the graph $G[V_t \cup \mathrm{bd}(\bar{V}_t)]$: We set the table entry $\mathcal{T}[t, (S, M, Q), i, j] = 1$, where $i \in [0..n]$ and $j \in [0..2]$, if and only if the following conditions are satisfied. For an illustration of the table indices, see Figure 2.

- **(i)** There is a set of induced paths $\mathcal{I}$ of total size i in $G[V_t \cup \mathrm{bd}(\bar{V}_t)]$ such that $\mathcal{I}$ has j degree one endpoints in G_t.
- **(ii)** $E(\mathcal{I}) \cap E(G_{t,\bar{t}}) = E(S)$.
- **(iii)** M is a minimal vertex cover of $G_{t,\bar{t}} - V(S)$ such that $V(\mathcal{I}) \cap M = \emptyset$.
- **(iv)** Let D denote the vertices in $S[V_t]$ that have degree one in S. Let $Q = (s_1, t_1), \ldots, (s_\ell, t_\ell)$ be a partition of all but j vertices of D into pairs, throughout the following called a

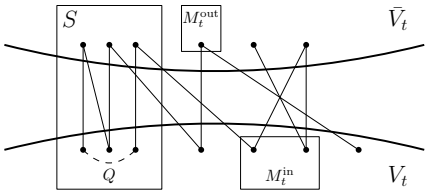

Figure 2 A crossing graph $G_{t,\bar{t}}$ and the structures associated with the table indices of the algorithm for Longest Induced Path. Note that by (1) and (4) it follows that if the table entry corresponding to the above structures is 1, then $j = 0$: Since both degree one endpoints in $S[V_t]$ are paired, the corresponding set of induced paths $\mathcal{I}$ has zero degree one endpoints in $G[V_t]$.

pairing, such that if we contract all edges in $\mathcal{I} - E(G_{t,\bar{t}})$ from $\mathcal{I}$ incident with at least one vertex not in S (we denote the resulting graph as $S \odot Q$) we obtain the same graph as when adding $\{s_k, t_k\}$ to S, for each $k \in [\ell]$.

Regarding (4), observe that $|D| = 2\ell + j$ and that there are j unpaired vertices in Q, each of which is connected to a degree one endpoint of $\mathcal{I}$ in G_t. For notational convenience, we will denote by $\mathcal{T}_t$ all table entries that have the node $t \in V(T)$ as the first index.

We now show that the solution to Longest Induced Path can be obtained from a table entry corresponding to the root r of T and hence ensure that the information stored in $\mathcal{T}$ is sufficient.

▶ **Proposition 5 (★).** *G contains an induced path of length i if and only if $\mathcal{T}[r, (\emptyset, \emptyset, \emptyset), i, 2] = 1$.*

Throughout the following, we denote by $\mathcal{S}_t$ the set of all sets of induced disjoint paths in $G_{t,\bar{t}}$ on at most $4w$ vertices (which includes all possible intersections of partial solutions with $G_{t,\bar{t}}$ by Lemma 4), for $S \in \mathcal{S}_t$ by $\mathcal{M}_{t,S}$ the set of all minimal vertex covers of $G_{t,\bar{t}} - V(S)$ and by $\mathcal{Q}_{t,S}$ the set of all pairings of degree one vertices in $S[\text{bd}(V_t)]$. We now argue that the number of such entries is bounded by a polynomial in n whose degree is $\mathcal{O}(w)$.

▶ **Proposition 6.** *For each $t \in V(T)$, there are at most $n^{\mathcal{O}(w)}$ table entries in $\mathcal{T}_t$ and they can be enumerated in time $n^{\mathcal{O}(w)}$.*

Proof. Note that each index is an element of $\mathcal{S}_t \times \mathcal{M}_{t,S} \times \mathcal{Q}_{t,S} \times [0..n] \times [0..2]$. Since the size of each maximum induced matching in $G_{t,\bar{t}}$ is at most w, we know by Lemma 4 that the size of each index S is bounded by $4w$, so $|\mathcal{S}_t| \leq \mathcal{O}(n^{4w})$. By the Minimal Vertex Covers Lemma (Corollary 3), $|\mathcal{M}_{t,S}| \leq n^{\mathcal{O}(w)}$. Since the number of vertices in S is bounded by $4w$, we know that $|\mathcal{Q}_{t,S}| \leq w^{\mathcal{O}(w)}$ and since $i \in [0..n]$ and $j \in [0..2]$, we can conclude that the number of table entries for each $t \in V(T)$ is at most $\mathcal{O}(n^{4w}) \cdot n^{\mathcal{O}(w)} \cdot w^{\mathcal{O}(w)} \cdot (n+1) \cdot 3 = n^{\mathcal{O}(w)}$. Clearly, all elements in $\mathcal{S}_t$ and $\mathcal{Q}_{t,S}$ can be enumerated in time $n^{\mathcal{O}(w)}$ and by the Minimal Vertex Covers Lemma, we know that all elements in $\mathcal{M}_{t,S}$ can be enumerated in time $n^{\mathcal{O}(w)}$ as well. The claimed time bound on the enumeration of the table indices follows. ◀

In the remainder of the proof we will describe how to fill the table entries from the leaves of T to its root, asserting the correctness of the updates in the table. Together with Proposition 5, this will yield the correctness of the algorithm. The description of how the tables are filled at a leaf node are deferred to the full version [10].

Internal nodes of T. Let $t \in V(T)$ be an internal node of T, let $(S, M, Q) \in \mathcal{S}_t \times \mathcal{M}_{t,S} \times \mathcal{Q}_{t,S}$, let $i \in [0..n]$ and $j \in [0..2]$. We show how to compute the table entry $\mathcal{T}[t, (S, M, Q), i, j]$ from

table entries corresponding to the children a and b of t in T. To do so, we have to take into account the ways in which partial solutions for $G[V_a \cup \mathrm{bd}(\bar{V_a})]$ and $G[V_b \cup \mathrm{bd}(\bar{V_b})]$ interact. We therefore try all pairs of indices $\mathfrak{I}_a = ((S_a, M_a, Q_a), i_a, j_a)$, $\mathfrak{I}_b = ((S_b, M_b, Q_b), i_b, j_b)$ and for each such pair, first check whether it is 'compatible' with $\mathfrak{I}_t$: We say that $\mathfrak{I}_a$ and $\mathfrak{I}_b$ are *compatible with* $\mathfrak{I}_t$ if and only if any partial solution $\mathcal{I}_a$ represented by $\mathfrak{I}_a$ for $G[V_a \cup \mathrm{bd}(\bar{V_a})]$ and $\mathcal{I}_b$ represented by $\mathfrak{I}_b$ for $G[V_b \cup \mathrm{bd}(\bar{V_b})]$ can be combined to a partial solution $\mathcal{I}_t$ for $G[V_t \cup \mathrm{bd}(\bar{V_t})]$ that is represented by the index $\mathfrak{I}_t$. We then set $\mathcal{T}_t[\mathfrak{I}_t] := 1$ if and only if we can find a compatible pair of indices $\mathfrak{I}_a$, $\mathfrak{I}_b$ as above such that $\mathcal{T}_a[\mathfrak{I}_a] = 1$ and $\mathcal{T}_b[\mathfrak{I}_b] = 1$.

Step 0 (Valid Index). We first check whether the index $\mathfrak{I}_t$ can represent a valid partial solution of $G[V_t \cup \mathrm{bd}(\bar{V_t})]$. The definition of the table entries requires that $S \odot Q$ is a disjoint union of paths, so if $S \odot Q$ is not a disjoint union of paths, we set $\mathcal{T}_t[\mathfrak{I}_t] := 0$ and skip the remaining steps. In general, the number of degree one vertices in $V(S \odot Q) \cap V_t$ has to be equal to j and we can proceed as described in Steps 1-4, except for the following special cases, the details of which can be found in the full version [10].

Special Case 1 ($j = 2$, $V(S \odot Q) \cap V_t$ has 0 deg. 1 vertices).
Special Case 2 ($j = 2$, $V(S \odot Q) \cap V_t$ has 2 deg. 1 vertices in same component).
Special Case 3 ($j = 0$ and $S = \emptyset$).

Step 1 (Induced disjoint unions of paths). We now check whether S_a and S_b are compatible with S. We have to ensure that $S \cap G_{a,\bar{t}} = S_a \cap G_{a,\bar{t}}$, $S \cap G_{b,\bar{t}} = S_b \cap G_{b,\bar{t}}$ and $S_a \cap G_{a,b} = S_b \cap G_{a,b}$. If these conditions are not satisfied, we skip the current pair of indices $\mathfrak{I}_a$, $\mathfrak{I}_b$. In the following, we use the notation $R = S_a \cap G_{a,b}$ $(= S_b \cap G_{a,b})$.

Step 2 (Pairings of degree one vertices and j). First, we deal with Special Case 1, i.e. $j = 2$ and $S = \emptyset$. We then check whether the graph obtained from taking R and adding an edge (and, if not already present, the corresponding vertices) for each pair in Q_a and Q_b is a single induced path. Note that we require the values of the integers j_a and j_b to be the number of endpoints of the resulting path in V_a and V_b, respectively.

Since the case $j = 0$ and $S = \emptyset$ is dealt with in Special Case 3 and S cannot be empty whenever $j = 1$, we may from now on assume that $S \neq \emptyset$ and hence $S \odot Q \neq \emptyset$.[1]

Consider the graph on vertex set $V(S) \cup V(R)$ whose edges consist of the edges in S and R together with the pairs in Q_a and Q_b. We then contract all edges in R and all edges that were added due to the pairings Q_a and Q_b and incident with a vertex not in S, and denote the resulting graph by $\mathcal{H}$. Then, Q_a and Q_b are compatible if and only if $\mathcal{H} = S \odot Q$. By the definition of the table entries (and since by Step 0, $S \odot Q$ is a disjoint union of paths) we can then see that Q_a, Q_b together with the edges of R connect the paired degree one vertices of Q as required. We furthermore need to ensure that the values of the integers j_a and j_b are the number of degree one endpoints in $\mathcal{H}[V_a]$ and $\mathcal{H}[V_b]$, respectively. For an illustration see Figure 3.

Step 3 (Minimal vertex covers). We now describe the checks we have to perform to ensure that M_a and M_b are compatible with M, which from now on we will denote by M_t to avoid confusion. For ease of exposition, we denote by $\mathcal{I}_t$, $\mathcal{I}_a$ and $\mathcal{I}_b$ (potential) partial solutions corresponding to $\mathfrak{I}_t$, $\mathfrak{I}_a$ and $\mathfrak{I}_b$, respectively.

Recall that the purpose of the minimal vertex cover M_t is to ensure that no unwanted edges appear between vertices used by the partial solution $\mathcal{I}_t$ and any partial solution of $G[\bar{V_t} \setminus \mathrm{bd}(\bar{V_t})]$ that can be combined with $\mathcal{I}_t$. Hence, when checking whether $\mathcal{I}_a$ and $\mathcal{I}_b$ can

[1] Note that it could still happen that $Q = \emptyset$ but since this does not essentially influence the following argument, we assume that $Q \neq \emptyset$.

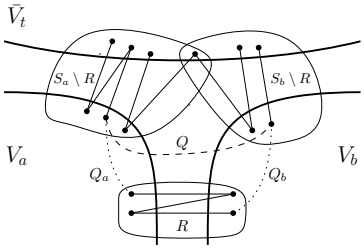

Figure 3 Step 3 of the join operation. Recall that $S_a \cap G_{a,b} = S_b \cap G_{a,b} = R$ by Step 1.

be combined to $\mathcal{I}_t$ without explicitly having access to these sets of induced disjoint paths, we have to make sure that the indices $\mathfrak{I}_a$ and $\mathfrak{I}_b$ assert the absence of unwanted edges — for any intersection of a partial solution with $G_{a,\bar{a}}$ and $G_{b,\bar{b}}$, as well as with $G_{a,b}$. Recall that $G_{a,\bar{a}} = G_{a,\bar{t}} \cup G_{a,b}$ and $G_{b,\bar{b}} = G_{b,\bar{t}} \cup G_{a,b}$.

We distinguish several cases, depending on where the unwanted edge might appear: First, between two intermediate vertices of partial solutions and second, between a vertex in S_a or S_b and an intermediate vertex. Step 3.1 handles the former and Step 3.2 the latter. In Step 3.1, we additionally have to distinguish whether the edge might appear in $G_{a,b}$ or in $G_{a,\bar{t}}$ (respectively, in $G_{b,\bar{t}}$). In the following, we let $M_a^{\text{out}(b)} := M_a^{\text{out}} \cap V_b$ and $M_b^{\text{out}(a)} := M_b^{\text{out}} \cap V_a$.

Step 3.1.1 (intermediate-intermediate, $G_{a,b}$). $M_a^{\text{out}(b)} \subseteq M_b^{\text{in}}$ and $M_b^{\text{out}(a)} \subseteq M_a^{\text{in}}$: A vertex $v \in M_a^{\text{out}(b)}$ can have a neighbor $w \in V_a$ which is used as an intermediate vertex in $\mathcal{I}_a$. Hence, to avoid that the unwanted edge $\{v, w\}$ appears in the combined solution $\mathcal{I}_a \cup \mathcal{I}_b$, we have to make sure that v is not used by $\mathcal{I}_b$, which is asserted if $v \in M_b^{\text{in}}$. By a symmetric argument we justify that $M_b^{\text{out}(a)} \subseteq M_a^{\text{in}}$.

Step 3.1.2 (intermediate-intermediate, $G_{a,\bar{t}}$ or $G_{b,\bar{t}}$). We have to check the following two conditions, the first one regarding M_t^{in} and the second one regarding M_t^{out}.

- **(a)** $M_t^{\text{in}} \subseteq M_a^{\text{in}} \cup M_b^{\text{in}}$: By the definition of M_t^{in}, $\mathcal{I}_t$ has to avoid the vertices in M_t^{in}. Hence, $\mathcal{I}_a$ and $\mathcal{I}_b$ have to avoid the vertices in M_t^{in} as well, which is ensured if for $v \in M_t^{\text{in}} \cap V_a$, we have that $v \in M_a^{\text{in}}$ and for $w \in M_t^{\text{in}} \cap V_b$, we have that $w \in M_b^{\text{in}}$.
- **(b)** For each vertex $v \in M_t^{\text{out}}$ having a neighbor x in V_a such that x is also contained in $V_a \setminus (V(S_a) \cup M_a^{\text{in}})$, we have that $v \in M_a^{\text{out}}$: Recall that by the definition of M_t^{out}, $\mathcal{I}_t$ could use the vertex x as an intermediate vertex. If $x \notin V(S_a) \cup M_a^{\text{in}}$, this means that x might be used by $\mathcal{I}_a$ as an intermediate vertex as well. Now, in a table entry representing a partial solution $\mathcal{I}_a$ using x, this is signalized by having $v \in M_a^{\text{out}}$. We check the analogous condition for M_b^{out}.

Step 3.2 (intermediate-(S_a or S_b)). $N_a[V(S_b) \setminus V(S_a)] \subseteq M_a^{\text{in}}$ and $N_b[V(S_a) \setminus V(S_b)] \subseteq M_b^{\text{in}}$: We justify the first condition and note that the second one can be argued for symmetrically. Clearly, $\mathcal{I}_a$ cannot have a neighbor x of any vertex $v \in V(S_b)$ as an intermediate vertex, if $\mathcal{I}_a$ is to be combined with $\mathcal{I}_b$. However, if $v \in V(S_a)$, then $\mathcal{I}_a$ does not use x by Part (2) of the definition of the table entries. Note that this includes all vertices in $V(R) \subseteq V(S_a)$. If on the other hand, $v \in V(S_b) \setminus V(S_a)$ then the neighbors of v have not been accounted for earlier, since v is not a vertex in the partial solution $\mathcal{I}_a$. Hence, we now have to assert that $\mathcal{I}_a$ does not use x, the neighbor of v, and so we require that $x \in M_a^{\text{in}}$.

Step 4 (i). We consider all pairs of integers i_a, i_b such that $i = i_a + i_b - |V(R)|$. By Step 2, all vertices in R are used in the partial solution $\mathcal{I}_t$. They are counted twice, since they are both accounted for in $\mathcal{I}_a$ and in $\mathcal{I}_b$.

Now, we let $\mathcal{T}_t[\mathfrak{I}_t] = 1$ if and only if there is a pair of indices $\mathfrak{I}_a = ((S_a, M_a, Q_a), i_a, j_a)$ and $\mathfrak{I}_b = ((S_b, M_b, Q_b), i_b, j_b)$ passing all checks performed in Steps 1-4 above, such that $\mathcal{T}_a[\mathfrak{I}_a] = 1$ and $\mathcal{T}_b[\mathfrak{I}_b] = 1$. This finishes the description of the algorithm.

▶ **Proposition 7 (★).** *Let $t \in V(T)$. The table entries $\mathcal{T}_t[\mathfrak{I}_t]$ computed according to Steps 0-4 above are correct.*

By Propositions 5 and 7 and the fact that in the leaf nodes of T, we enumerate all possible partial solutions, we know that the algorithm we described is correct. Since by Proposition 6, there are at most $n^{\mathcal{O}(w)}$ table entries at each node of T (and they can be enumerated in time $n^{\mathcal{O}(w)}$), the value of each table entry in $\mathcal{T}_t$ as above can be computed in time $n^{\mathcal{O}(w)} \cdot n^{\mathcal{O}(w)} \cdot n^{\mathcal{O}(1)} = n^{\mathcal{O}(w)}$, since each check described in Steps 0-4 can be done in time polynomial in n. Since additionally, $|V(T)| = \mathcal{O}(n)$, the total runtime of the algorithm is $n^{\mathcal{O}(w)} \cdot n^{\mathcal{O}(w)} \cdot \mathcal{O}(n) = n^{\mathcal{O}(w)}$ and we have the following theorem.

▶ **Theorem 8.** *There is an algorithm that given a graph G on n vertices and a branch decomposition $(T, \mathcal{L})$ of G, solves* Longest Induced Path *in time $n^{\mathcal{O}(w)}$, where w denotes the mim-width of $(T, \mathcal{L})$.*

3.2 Induced Disjoint Paths

In this section, we build upon the ideas of the algorithm for Longest Induced Path presented above to obtain an $n^{\mathcal{O}(w)}$-time algorithm for the following parameterized problem.

Induced Disjoint Paths (IDP)/Mim-Width
Input: A graph G with branch decomposition $(T, \mathcal{L})$ and pairs of vertices (x_1, y_1), …, (x_k, y_k) of G.
Parameter: $w := \mathrm{mimw}(T, \mathcal{L})$
Question: Does G contain a set of vertex-disjoint induced paths $P_1, \ldots, P_k$, such that for $i \in [k]$, P_i is a path from x_i to y_i and for $i \neq j$, P_i does not contain a vertex adjacent to a vertex in P_j?

Throughout the remainder of this section, we refer to the vertices $\{x_i, y_i\}$, where $i \in [k]$ as the *terminals* and we denote the set of all terminals by $X := \bigcup_{i \in [k]} \{x_i, y_i\}$. We furthermore use the following notation: We denote by $\mathcal{C}(G)$ the set of all connected components of G and for a vertex $v \in V(G)$, $C_G(v)$ refers to the connected component containing v.

We observe how a solution $\mathcal{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_k)$ interacts with the graph $G[V_t \cup \mathrm{bd}(\bar{V}_t)]$, for some $t \in V(T)$. In this case, for each $i \in [k]$, $\mathcal{P}_i$ is an (x_i, y_i)-path and additionally for $j \neq i$, there is no vertex in $\mathcal{P}_i$ adjacent to a vertex of $\mathcal{P}_j$. The intersection of $\mathcal{P}$ with $G[V_t \cup \mathrm{bd}(\bar{V}_t)]$ is a subgraph $\mathcal{I} = (\mathcal{I}_1, \ldots, \mathcal{I}_k)$, where each $\mathcal{I}_i$ is a (possibly empty) induced disjoint union of paths which is the intersection of the (x_i, y_i)-path $\mathcal{P}_i$ with $G[V_t \cup \mathrm{bd}(\bar{V}_t)]$. Note that each terminal $v_i \in \{x_i, y_i\}$ that is contained in $V_t \cup \mathrm{bd}(\bar{V}_t)$ is also contained in $V(\mathcal{I}_i)$.

Again our goal is to bound the number of table entries at each node $t \in V(T)$ by $n^{\mathcal{O}(w)}$, so we focus on the intersection of $\mathcal{I}$ with the crossing graph $G_{t,\bar{t}}$. There are several reasons why $\mathcal{I}_i$ can have a nonempty intersection with the crossing graph $G_{t,\bar{t}}$: If precisely one of x_i and y_i is contained in V_t, then the path $\mathcal{P}_i$ must cross the boundary of G_t. If both x_i and y_i are contained in V_t ($\bar{V}_t$), yet $\mathcal{P}_i$ uses a vertex of $\bar{V}_t$ (V_t), then it crosses the boundary of G_t.

We now turn to the definition of the table indices. Let us first point out what table indices in the resulting algorithm for Induced Disjoint Paths have in common with the indices in the algorithm for Longest Induced Path and we refer to Section 3.1 for the motivation and details. These similarities arise since in both problems, the intersection of a solution with a crossing graph $G_{t,\bar{t}}$ is an induced disjoint union of paths.

- The intersection of $\mathcal{I}$ with the edges of $G_{t,\bar{t}}$ is S, an induced disjoint union of paths where each component contains at least one edge.
- M is a minimal vertex cover of $G_{t,\bar{t}} - V(S)$ such that $M \cap V(S) = \emptyset$.

The first important observation to be made is that by Lemma 4, the number of components of S is linearly bounded in w and hence at most $\mathcal{O}(w)$ paths of $\mathcal{P}$ can have a nonempty intersection with $G_{t,\bar{t}}$. We need to store information about which path $\mathcal{P}_i$ (resp., to which $\mathcal{I}_i$) the components of S correspond to. To do so, another part of the index will be a labeling function $\lambda\colon \mathcal{C}(S) \to [k]$, whose purpose is to indicate that each component $C \in \mathcal{C}(S)$ is contained in $\mathcal{I}_{\lambda(C)}$. We just observed that each such λ contains at most $\mathcal{O}(w)$ entries.

Let $i \in [k]$. Again, we need to indicate how (some of) the components of S are connected via $\mathcal{I}_i$ in $G[V_t]$. As before, we do so by considering a pairing of the vertices in $S[V_t]$ that have degree one in S, however in this case we also have to take into account the labeling function λ. That is, two such vertices s and t can only be paired if they belong to the same induced disjoint union of paths $\mathcal{I}_i$.

In accordance with the above discussion, we define the table entries as follows. We let $\mathcal{T}[t, (S, M, \lambda, Q_\lambda)] = 1$ if and only if the following conditions are satisfied.

- **(i)** There is an induced disjoint union of paths $\mathcal{I} = (\mathcal{I}_1, \ldots, \mathcal{I}_k)$ in $G[V_t \cup \mathrm{bd}(\bar{V_t})]$, such that for $i \neq j$, $\mathcal{I}_i$ does not contain a vertex adjacent (in G) to a vertex in $\mathcal{I}_j$. For each $i \in [k]$, we have that if $v_i \in \{x_i, y_i\} \cap V_t$, then $v_i \in \mathcal{I}_i$. Furthermore, v_i has degree one in $\mathcal{I}_i$.
- **(ii)** **(a)** $E(\mathcal{I}) \cap E(G_{t,\bar{t}}) = E(S)$.
  - **(b)** $\lambda\colon \mathcal{C}(S) \to [k]$ is a labeling function of the connected components of S, such that for each component $C \in \mathcal{C}(S)$, $\lambda(C) = i$ if and only if $C \subseteq \mathcal{I}_i$.
- **(iii)** M is a minimal vertex cover of $G_{t,\bar{t}} - V(S)$ such that $V(\mathcal{I}) \cap M = \emptyset$.
- **(iv)** Let D denote the set of vertices in $S[V_t]$ that have degree one in S and let $X_t = X \cap V_t$. Then, Q_λ is a pairing (i.e. a partition into pairs) of the vertices in $D \triangle X_t$ with the following properties.[2]
  1. $(s, t) \in Q_\lambda$ if and only if there is a path from s to t in $\mathcal{I}[V_t]$. Note that this implies that $(s, t) \in Q_\lambda$ only if both s and t belong to the same $\mathcal{I}_i$ for some $i \in [k]$ and in particular only if $s, t \in V(\lambda^{-1}(i)) \cup \{x_i, y_i\}$.
  2. For each $i \in [k]$, $(x_i, y_i) \in Q_\lambda$ only if $\lambda^{-1}(i) = \emptyset$, i.e. no component of S has label i.

Using this definition of the table indices, one can design an algorithm solving Induced Disjoint Paths in time $n^{\mathcal{O}(w)}$ analogously to the algorithm for Longest Induced Path. We give further details in the full version [10] and we have the following theorem.

▶ **Theorem 9.** *There is an algorithm that given a graph G on n vertices, pairs of terminal vertices $(x_1, y_1), \ldots, (x_k, y_k)$ and a branch decomposition $(T, \mathcal{L})$ of G, solves* Induced Disjoint Paths *in time $n^{\mathcal{O}(w)}$, where w denotes the mim-width of $(T, \mathcal{L})$.*

3.3 H-Induced Topological Minor

Let G be a graph and $uv \in E(G)$. We call the operation of replacing the edge uv by a new vertex x and edges ux and xv the *edge subdivision* of uv. We call a graph H a *subdivision* of G if it can be obtained from G by a series of edge subdivisions. We call H an *induced topological minor* of G if a subdivision of H is isomorphic to an induced subgraph of G.

[2] We denote by '$\triangle$' the symmetric difference, i.e. $D \triangle X_t = (D \cup X_t) \setminus (D \cap X_t)$. Q_λ is a pairing on $D \triangle X_t$, since if a terminal v_i is contained in D, it is supposed to be paired 'with itself': Since v_i has degree one in $\mathcal{I}_i$ by (1) and is incident to an edge in S, v_i cannot be paired with another vertex.

H-Induced Topological Minor/Mim-Width
Input: A graph G with branch decomposition $(T, \mathcal{L})$
Parameter: $w := \text{mimw}(T, \mathcal{L})$
Question: Does G contain H as an induced topological minor?

▶ **Theorem 10 (★).** *There is an algorithm that given a graph G on n vertices and a branch decomposition $(T, \mathcal{L})$ of G, solves H-Induced Topological Minor in time $n^{\mathcal{O}(w)}$, where H is a fixed graph and w the mim-width of $(T, \mathcal{L})$.*

References

1 Rémy Belmonte, Petr A. Golovach, Pinar Heggernes, Pim van 't Hof, Marcin Kaminski, and Daniël Paulusma. Detecting fixed patterns in chordal graphs in polynomial time. *Algorithmica*, 69(3):501–521, 2014. doi:10.1007/s00453-013-9748-5.

2 Rémy Belmonte and Martin Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theor. Comput. Sci.*, 511:54–65, 2013. doi:10.1016/j.tcs.2013.01.011.

3 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theoret. Comput. Sci.*, 511:66–76, 2013.

4 Jirí Fiala, Marcin Kaminski, Bernard Lidický, and Daniël Paulusma. The k-in-a-path problem for claw-free graphs. *Algorithmica*, 62(1-2):499–519, 2012. doi:10.1007/s00453-010-9468-z.

5 Fanica Gavril. Algorithms for maximum weight induced paths. *Inf. Process. Lett.*, 81(4):203–208, 2002. doi:10.1016/S0020-0190(01)00222-8.

6 Petr A. Golovach, Daniël Paulusma, and Erik Jan van Leeuwen. Induced disjoint paths in at-free graphs. In Fedor V. Fomin and Petteri Kaski, editors, *Algorithm Theory - SWAT 2012 - 13th Scandinavian Symposium and Workshops, Helsinki, Finland, July 4-6, 2012. Proceedings*, volume 7357 of *Lecture Notes in Computer Science*, pages 153–164. Springer, 2012. doi:10.1007/978-3-642-31155-0_14.

7 Petr A. Golovach, Daniël Paulusma, and Erik Jan van Leeuwen. Induced disjoint paths in circular-arc graphs in linear time. *Theor. Comput. Sci.*, 640:70–83, 2016. doi:10.1016/j.tcs.2016.06.003.

8 Petr Hlinený, Sang-il Oum, Detlef Seese, and Georg Gottlob. Width parameters beyond tree-width and their applications. *Comput. J.*, 51(3):326–362, 2008. doi:10.1093/comjnl/bxm052.

9 Tetsuya Ishizeki, Yota Otachi, and Koichi Yamazaki. An improved algorithm for the longest induced path problem on k-chordal graphs. *Discrete Applied Mathematics*, 156(15):3057–3059, 2008. doi:10.1016/j.dam.2008.01.019.

10 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Polynomial-time algorithms for the longest induced path and induced disjoint paths problems on graphs of bounded mim-width. *ArXiv e-prints*, 2017. arXiv:1708.04536.

11 Dong Yeap Kang, O-joung Kwon, Torstein J. F. Strømme, and Jan Arne Telle. A width parameter useful for chordal and co-comparability graphs. In Sheung-Hung Poon, Md. Saidur Rahman, and Hsu-Chun Yen, editors, *WALCOM: Algorithms and Computation, 11th International Conference and Workshops, WALCOM 2017, Hsinchu, Taiwan, March 29-31, 2017, Proceedings.*, volume 10167 of *Lecture Notes in Computer Science*, pages 93–105. Springer, 2017. doi:10.1007/978-3-319-53925-6_8.

12 Ken-ichi Kawarabayashi and Yusuke Kobayashi. The induced disjoint paths problem. In Andrea Lodi, Alessandro Panconesi, and Giovanni Rinaldi, editors, *Integer Programming and Combinatorial Optimization, 13th International Conference, IPCO 2008, Bertinoro,*

Italy, May 26-28, 2008, Proceedings, volume 5035 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2008. doi:10.1007/978-3-540-68891-4_4.

13 Dieter Kratsch, Haiko Müller, and Ioan Todinca. Feedback vertex set and longest induced path on at-free graphs. In Hans L. Bodlaender, editor, *Graph-Theoretic Concepts in Computer Science, 29th International Workshop, WG 2003, Elspeet, The Netherlands, June 19-21, 2003, Revised Papers*, volume 2880 of *Lecture Notes in Computer Science*, pages 309–321. Springer, 2003. doi:10.1007/978-3-540-39890-5_27.

14 Sridhar Natarajan and Alan P. Sprague. Disjoint paths in circular arc graphs. *Nordic J. of Computing*, 3(3):256–270, 1996. URL: http://dl.acm.org/citation.cfm?id=642150.642154.

15 Neil Robertson and Paul D. Seymour. Graph minors .xiii. the disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.

16 Sigve Hortemo Sæther and Martin Vatshelle. Hardness of computing width parameters based on branch decompositions over the vertex set. *Theor. Comput. Sci.*, 615:120–125, 2016.

17 Jan Arne Telle and Andrzej Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM J. Discrete Math.*, 10(4):529–550, 1997. doi:10.1137/S0895480194275825.

18 Martin Vatshelle. *New Width Parameters of Graphs*. PhD thesis, University of Bergen, 2012.