Mim-Width I. Induced Path Problems*

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

[1]Department of Informatics, University of Bergen, Norway.
{lars.jaffke, jan.arne.telle}@uib.no
[2]Department of Mathematics, Incheon National University, South Korea.
ojoungkwon@gmail.com

July 15, 2019

Abstract

We initialize a series of papers deepening the understanding of algorithmic properties of the width parameter *maximum induced matching width* (mim-width) of graphs. In this first volume we provide the first polynomial-time algorithms on graphs of bounded 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 (given a k-trapezoid model/circular k-trapezoid model), k-Polygon, Dilworth-k and Co-k-Degenerate graphs for fixed k. We contrast these positive results to the fact that problems about finding long *non-induced* paths remain hard on graphs of bounded mim-width: We show that Hamiltonian Cycle (and hence Hamiltonian Path) is NP-hard on graphs of linear mim-width 1; this further hints at the expressive power of the mim-width parameter.

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. [12] for an excellent survey and motivation for width-parameters of graphs. In 2012, Vatshelle [30] defined the

*The work was partially done while the authors were at Polytechnic University of Valencia, Spain. The paper is based on extended abstracts that appeared in IPEC 2017 [16] and STACS 2018 [17]. Published in *Discrete Applied Mathematics*.

[†]Supported by the Bergen Research Foundation (BFS).

[‡]Supported by the National Research Foundation of Korea (NRF) grant funded by the Ministry of Education (No. NRF-2018R1D1A1B07050294), and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC consolidator grant DISTRUCT, agreement No. 648527). Part of the research took place while Kwon was at Logic and Semantics, Technische Universität Berlin, Berlin, Germany.

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 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, 30], 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[1] 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 [29] 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 mainly study problems related to finding *induced* paths in graphs, namely Longest Induced Path, Induced Disjoint Paths and H-Induced Topological Minor, all having a connectivity constraint that is not locally checkable. 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 [9] 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. [13]), while Kratsch et al. [23] solved the problem on AT-free graphs in polynomial time. Kang et al. [20] 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 [27], while if k is part of the input it is NP-complete on graphs of linear mim-width 1 (Interval graphs) [25]. In contrast, Induced Disjoint Paths is NP-complete already for $k = 2$ paths [21]. In this paper we consider the number of paths k as part of the input. In this setting Induced Disjoint Paths is NP-complete on claw-free graphs, as shown by Fiala et al. [8], while Golovach et al. [11] 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. [10].

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

[1] I.e. in $f(w) \cdot n^{g(w)}$ time where w denotes the mim-width of the input graphs and f and g are computable functions.

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,[2] k-Polygon, Dilworth-k and Co-k-Degenerate graphs for fixed k, all graph classes of bounded mim-width [2].[3]

The problem of computing the mim-width of general graphs was shown to be W[1]-hard [28], where mim-width itself is the parameter, 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 [28].

We contrast the above mentioned positive results with a proof that Hamiltonian Cycle is NP-complete on graphs of linear mim-width 1, even when given a decomposition. Panda and Pradhan [26] showed that Hamiltonian Cycle is NP-complete on Rooted Directed Path graphs and we show that the graphs constructed in their reduction have linear mim-width 1. This provides evidence that the class of graphs of linear mim-width 1 is larger than one might have previously expected. Up until now, on all graph classes of linear mim-width 1, Hamiltonian Cycle was known to be polynomial time (Permutation [5]), or even linear time (Interval [22], Convex [24]) solvable.

Let us outline the above mentioned algorithmic results. 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 the following. Let (A, B) be a cut induced by an edge of a given branch decomposition of mim-width w of an input graph. It is sufficient to consider induced subgraphs on at most $\mathcal{O}(w)$ vertices as intersections of solutions with the edges crossing (A, B). For instance, in the Longest Induced Paths problem, an induced path is a target solution, and Figure 1 describes a situation where an induced path crosses a cut. 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 an induced disjoint union of paths. Thus, we enumerate all subgraphs of size at most $\mathcal{O}(w)$, which are induced 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 an 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. This is achieved 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 covers 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 induced 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

[2] Given a k-trapezoid/circular k-trapezoid model.

[3] In [2], results are stated in terms of d-neighborhood equivalence, but in the proof, they actually gave a bound on mim-width or linear mim-width. Note that Interval, Permutation and k-Trapezoid graphs are AT-free, so polynomial-time algorithms for all three problems were already known [10].

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. This technique was also crucially employed in later work, in particular in the algorithm that solves Feedback Vertex Set in $n^{\mathcal{O}(w)}$ time in the same parameterization [18].

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, 10]) but we include it here for completeness.

2 Preliminaries

We assume the reader to be familiar with the basic notions in graph theory and parameterized complexity and refer to [4, 6, 7] for an introduction.

For integers a and b with $a \leq b$, we let $[a..b] \coloneqq \{a, a+1, \ldots, b\}$ and if a is positive, we define $[a] \coloneqq [1..a]$. 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. $G[X] \coloneqq \left(X, E(G) \cap \binom{X}{2}\right)$. If $H \subseteq G$ and $X \subseteq V(G)$ then we let $H[X] \coloneqq H[X \cap V(H)]$. 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 and a cut is called *nontrivial* if $A \neq \emptyset$ and $B \neq \emptyset$. A graph G is *connected* if for each nontrivial cut (A, B) of $V(G)$, there is an edge $\{a, b\} \in E(G)$ with $a \in A$ and $b \in B$. A *connected component* of a graph G is a maximal connected subgraph of G.

For a vertex $v \in V(G)$, we denote by $N(v)$ the set of *neighbors* of v in G, i.e. $N(v) \coloneqq \{u \in V(G) \mid \{v, u\} \in E(G)\}$. The *degree* of a vertex $v \in V(G)$ is the number of its neighbors, i.e. $\deg(v) \coloneqq |N(v)|$. For a set of vertices $X \subseteq V(G)$, we furthermore define $N(X) \coloneqq \bigcup_{x \in X} N(x) \setminus X$.

A connected graph all of whose vertices have degree two is called a *cycle*, a graph that does not contain a cycle as a subgraph is called a *forest*, a connected forest is a *tree*. A tree of maximum degree two is called a *path* and we call its unique two vertices of degree one its *endpoints*. We call the *length* of a path the number of its vertices. Let G be a graph and $P \subseteq G$ a path. Then, P is called an *induced path*, if $G[V(P)] = P$, i.e. if there are no additional edges in the subgraph induced by the vertex set of P. Similarly, if $I \subseteq G$ is a disjoint union of paths, we call I an *induced disjoint union of paths* if $G[V(I)] = I$.

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 $e = \{u, v\} \in E(G)$, the operation of *contracting* e is to remove the edge e from G and identifying u and v.

2.1 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 $\mathcal{L}$. For a vertex set $A \subseteq V(G)$, we let $\mathrm{mim}(A)$ denote the maximum size of an induced matching in $G[A, V(G) \setminus A]$. The *mim-width of* $(T, \mathcal{L})$, denoted by $\mathrm{mimw}(T, \mathcal{L})$, is defined as $\max_{e \in E(T)} \mathrm{mim}(A_1^e)$. 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 V_t the set of vertices that are mapped to a leaf that is a descendant of t, i.e. $V_t \coloneqq \{v \in V(G) \mid \mathcal{L}^{-1}(t') = v \text{ where } t' \text{ is a leaf descendant of } t \text{ in } T\}$. We let $\overline{V_t} \coloneqq V(G) \setminus V_t$ and $G_t \coloneqq G[V_t]$.

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. $\mathrm{bd}_B(A) \coloneqq \{v \in A \mid N(v) \cap B \neq \emptyset\}$. We define $\mathrm{bd}(A) \coloneqq \mathrm{bd}_{V(G)\setminus A}(A)$ 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 $G_{A,B} \coloneqq G[\mathrm{bd}_B(A), \mathrm{bd}_A(B)]$ 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 $G_{t_1,t_2} \coloneqq G_{V_{t_1}, V_{t_2}}$. We use the analogous shorthand notations $G_{t_1,\overline{t_2}} \coloneqq G_{V_{t_1}, \overline{V_{t_2}}}$ and $G_{\overline{t_1},t_2} \coloneqq G_{\overline{V_{t_1}}, V_{t_2}}$ (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*.

2.2 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.

Notice that the bound on the number can be easily obtained by combining two results, [2, Lemma 1] and [30, Theorem 3.5.5]; however, an enumeration algorithm is not given explicitly. To be self-contained, we state and prove it here.

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 $n^{\mathcal{O}(\mathrm{mim}(A))}$.*

Proof. Let $w := \mathrm{mim}(A)$. For each vertex set $R \subseteq A$ with $|R| \leq w$, let $X_R \subseteq A$ be the set of all vertices having a neighbor in $B \setminus N(R)$. We enumerate the sets in

$$\mathcal{M} = \{N(R) \cup X_R : R \subseteq A, |R| \leq w\}.$$

Clearly, we can enumerate them in time $n^{\mathcal{O}(w)}$. It is not difficult to see that each set in $\mathcal{M}$ is a minimal vertex cover. We claim that $\mathcal{M}$ is the set of all minimal vertex covers in H.

We use the result by Belmonte and Vatshelle [2, Lemma 1] that for a graph G and $A \subseteq V(G)$, $\mathrm{mim}(A) \leq k$ if and only if for every $S \subseteq A$, there exists $R \subseteq S$ such that $N(R) \cap (V(G) \setminus A) = N(S) \cap (V(G) \setminus A)$ and $|R| \leq k$.

Let U be a minimal vertex cover of H. Clearly, every vertex in $A \setminus U$ has no neighbors in $B \setminus U$, as U is a vertex cover. Therefore, by the result of Belmonte and Vatshelle, there exists $R \subseteq A \setminus U$ such that $|R| \leq w$ and $N(R) \cap B = N(A \setminus U) \cap B = U \cap B$. Clearly, $U \cap A = X_R$; if a vertex in $U \cap A$ has no neighbors in $B \setminus U$, then we can remove it from the vertex cover. Therefore, $U \in \mathcal{M}$, as required. □

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 $(T, \mathcal{L})$, starting at the leaves of T. 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

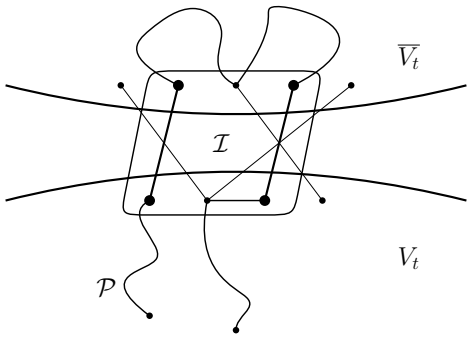

Figure 1: The intersection of an induced path $\mathcal{P}$ with $G[V_t \cup \mathrm{bd}(\overline{V_t})]$, which is an induced disjoint union of paths $\mathcal{I}$. The subgraph S to be used as an index for the corresponding table entry consists of the boldface vertices and edges in $\mathcal{I}$.

$G_{A,V(G)\setminus A}$ 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. 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. □

Remark 5. Unless stated otherwise, any path P (or equivalently, a component of a disjoint union of paths) we refer to throughout the remainder of this section is considered to be nontrivial, i.e. P contains at least one edge.

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 $G[V_t \cup \mathrm{bd}(\overline{V_t})]$, for some $t \in V(T)$. The intersection of $\mathcal{P}$ with $G[V_t \cup \mathrm{bd}(\overline{V_t})]$ 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.

Throughout the following, we call vertices of $\mathcal{I}$ that are *not* contained in $V(S)$ *intermediate vertices* w.r.t. the cut $(V_t, \overline{V_t})$. If the cut is clear from the context, we simply call them intermediate vertices. Note that by definition there cannot be any edges between intermediate vertices on opposite sides of the boundary. 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 $M \cap \overline{V_t}$. We therefore define the following two sets.

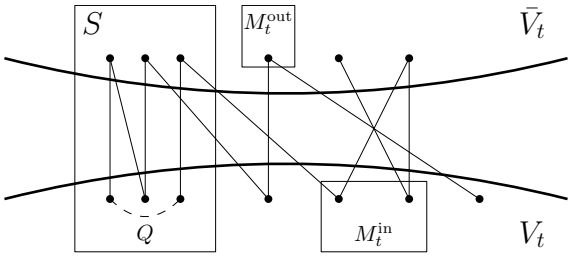

Figure 2: A crossing graph $G_{t,\bar{t}}$ and the structures associated with the table indices of the algorithm for Longest Induced Path. Note that by (i) and (iv) it follows that if the table entry corresponding to the above structures is 1, then $j = 0$: Since both degree one endpoints in $S[V_t]$ are paired, this means that the corresponding set of induced paths $\mathcal{I}$ has no degree one endpoints in $G[V_t]$.

- $M_t^{\text{in}} := M \cap V_t$ is the set of vertices that must be avoided by $\mathcal{I}$.
- $M_t^{\text{out}} := M \cap \overline{V_t}$ is the set of vertices that must be avoided by a partial solution (e.g. the intersection of $\mathcal{P}$ with $G[\overline{V_t}]$) 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 $\text{bd}(V_t)$). 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 $G[V_t \cup \text{bd}(\overline{V_t})]$: 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 $G[V_t \cup \text{bd}(\overline{V_t})]$ 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 $V(\mathcal{I}) \cap M = \emptyset$. (Recall that $M = M_t^{\text{in}} \cup M_t^{\text{out}}$.)

(iv) Let D denote the set of 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 *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 (iv), 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 6. *G contains an induced path of length i if and only if $\mathcal{T}[r, (\emptyset, \emptyset, \emptyset), i, 2] = 1$.*

Proof. Suppose G contains an induced path $\mathcal{P}$ of length i and consider the root r of $(T, \mathcal{L})$. Clearly, $G[V_r \cup \mathrm{bd}(\overline{V_r})] = G$, since $V_r = V(G)$ and $\mathrm{bd}(\overline{V_r}) = \emptyset$. Together with the fact that $\mathcal{P}$ is a path (and hence has two degree one endpoints in $G_r = G$, it follows that $\mathcal{P}$ satisfies Condition (i) for a table entry to be set to 1. Since $\mathrm{bd}(\overline{V_r}) = \emptyset$, it follows that for any index in $\mathcal{T}_r$, $S = \emptyset$, $M = \emptyset$ and $Q = \emptyset$. We can conclude that $\mathcal{T}[r, (\emptyset, \emptyset, \emptyset), i, 2] = 1$.

Now suppose $\mathcal{T}[r, (\emptyset, \emptyset, \emptyset), i, 2] = 1$. Then there is a set of induced paths $\mathcal{I}$ of total size i in G by (i) having two degree one endpoints in $G_r = G$. The latter allows us to conclude that $\mathcal{I}$ is in fact a single path. □

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 without isolated vertices (which includes all possible intersections of partial solutions with $G_{t,\bar{t}}$ by Lemma 4), and for $S \in \mathcal{S}_t$ denote 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 $S[\mathrm{bd}(V_t)]$. We now argue that the number of such entries is bounded by a polynomial in n whose degree is $\mathcal{O}(w)$.

Proposition 7. *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

$$\mathcal{O}(n^{4w}) \cdot n^{\mathcal{O}(w)} \cdot w^{\mathcal{O}(w)} \cdot (n+1) \cdot 3 = n^{\mathcal{O}(w)},$$

as claimed. 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 6, this will yield the correctness of the algorithm.

Leaves of T. Let $t \in V(T)$ be a leaf node of T and let $v = \mathcal{L}^{-1}(t)$. We observe that any nonempty intersection of an induced disjoint union of paths in $G_{t,\bar{t}}$ is a single edge (whose length/size is two) using v or a path on two edges (whose length/size is three) whose middle vertex is v. Hence, all indices in $\mathcal{T}_t$ that are set to 1 (with nonempty S) have the following properties: Either S is a single edge using v, and we denote the set of all such edges by $\mathcal{S}^1_{t,v}$ or a path on two edges with v as the middle vertex, and we denote the corresponding set of such paths as $\mathcal{S}^2_{t,v}$. For each $S \in \mathcal{S}^1_{t,v} \cup \mathcal{S}^2_{t,v}$, the corresponding minimal vertex cover of $G_{t,\bar{t}} - V(S)$ is empty, since $v \in V(S)$ and no edges remain in $G_{t,\bar{t}}$ when we remove v. Since v is the only vertex in G_t, $Q = \emptyset$ in both of these cases. If $S \in \mathcal{S}^1_{t,v}$, then $j = 1$ and if $S \in \mathcal{S}^2_{t,v}$ then $j = 0$. Additionally, a table entry is set to 1 if $S = \emptyset$ and $i = 0$ and since the solution is empty in this case, $Q = \emptyset$ and $j = 0$. The two corresponding minimal vertex

covers are $\{v\}$ and $N(v)$. Hence, we set the table entries in the leaves as follows.

$$\mathcal{T}[t,(S,M,Q),i,j] = \begin{cases} 1, & \text{if } S \in \mathcal{S}^1_{t,v}, M = \emptyset, Q = \emptyset, i = 2 \text{ and } j = 1 \\ 1, & \text{if } S \in \mathcal{S}^2_{t,v}, M = \emptyset, Q = \emptyset, i = 3 \text{ and } j = 0 \\ 1, & \text{if } S = \emptyset, M \in \{\{v\}, N(v)\}, Q = \emptyset, i = 0 \text{ and } j = 0 \\ 0, & \text{otherwise} \end{cases}$$

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 $G[V_a \cup \mathrm{bd}(\overline{V_a})]$ and $G[V_b \cup \mathrm{bd}(\overline{V_b})]$ 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 $G[V_a \cup \mathrm{bd}(\overline{V_a})]$ and $\mathcal{I}_b$ represented by $\mathfrak{I}_b$ for $G[V_b \cup \mathrm{bd}(\overline{V_b})]$ can be combined to a partial solution $\mathcal{I}_t$ for $G[V_t \cup \mathrm{bd}(\overline{V_t})]$ 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 $G[V_t \cup \mathrm{bd}(\overline{V_t})]$. 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.

Special Case 1 ($j = 2$, $V(S \odot Q) \cap V_t$ has 0 vertices of deg. 1 in $S \odot Q$). This is the case when $\mathcal{I}$ does not contain any edge in $E(G_{t,\bar{t}})$. It immediately follows that S has to be empty (and hence Q has to be empty). If not, we set $\mathcal{T}_t[\mathfrak{I}_t] = 0$ and skip the remaining steps. We would like to remark that the case when $j = 2$ and $S = \emptyset$ will have to be dealt with separately in Step 2, since $S \odot Q$ is the empty graph.

Special Case 2 ($j = 2$, $S \odot Q$ has a component C that is a path with 2 endpoints in V_t). In this case, $S \odot Q$ has to consist of a single component (and $\mathcal{I}$ of a single path): If there was another component C' in $S \odot Q$, C and C' could never be joined together to become a single path in the vertices of $\overline{V_t}$ and hence we can never obtain a valid solution from the partial solution represented by this index. So if $S \odot Q$ has more than one component, we set $\mathcal{T}_t[\mathfrak{I}_t] = 0$ and skip the remaining steps, otherwise we do not have to pay any further attention to this case in the following computations.

Special Case 3 ($j = 0$ and $S = \emptyset$). This is the case when $\mathcal{I}$ does not use any vertex of V_t. We then set $\mathcal{T}_t[\mathfrak{I}_t] = 1$ if and only if $i = 0$ and skip the remaining steps.

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
- $S_a \cap G_{a,b} = S_b \cap G_{a,b}$.

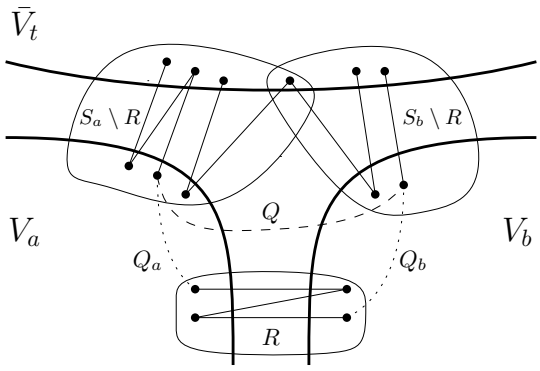

Figure 3: Step 2 of the join operation. Recall that $S_a \cap G_{a,b} = S_b \cap G_{a,b} = R$ by Step 1.

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 in $S \odot Q$ 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 $S \odot Q \neq \emptyset$.[4]

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 $G[\overline{V_t} \setminus \mathrm{bd}(\overline{V_t})]$ that can be combined with $\mathcal{I}_t$. Hence, when checking whether $\mathcal{I}_a$ and $\mathcal{I}_b$ can 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

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

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, $G_{a,b}$). We have to check that $M_a^{\text{out}(b)} \subseteq M_b^{\text{in}}$ and $M_b^{\text{out}(a)} \subseteq M_a^{\text{in}}$.

A vertex $v \in M_a^{\text{out}(b)}$ can have a neighbor $u \in V_a$ which is used as an intermediate vertex in $\mathcal{I}_a$. Hence, to avoid that the unwanted edge $\{u, v\}$ 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 $u \in M_t^{\text{in}} \cap V_b$, we have that $u \in M_b^{\text{in}}$.

(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)). We have to check that $V_a \cap N(V(S_b) \setminus V(S_a)) \subseteq M_a^{\text{in}}$ and $V_b \cap N(V(S_a) \setminus V(S_b)) \subseteq M_b^{\text{in}}$.

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 (ii) 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 8. *Let $t \in V(T)$. The table entries $\mathcal{T}_t[\mathfrak{I}_t]$ computed according to Steps 0-4 above are correct.*

Proof. Suppose there is partial solution $\mathcal{I}_t$, an induced disjoint union of paths in $G[V_t \cup \mathrm{bd}(\overline{V_t})]$. We claim that for any index $\mathfrak{I}_t$ representing the partial solution $\mathcal{I}_t$, $\mathcal{T}_t[\mathfrak{I}_t] = 1$ after the join operation described by Steps 0-4, assuming as the induction hypothesis that the table values of the children a and b of t are computed correctly. Any index $\mathfrak{I}_t$ representing $\mathcal{I}_t$ has the following properties: $S = \mathcal{I}_t \cap G_{t,\bar{t}}$ and the pairing Q of the degree one vertices in $S[V_t]$ can be obtained by letting $(s,t) \in Q$ if and only if there is a path connecting s and t in $\mathcal{I}_t[V_t]$. We furthermore have that $i = |V(\mathcal{I}_t)|$ and j is the number of degree one endpoints in $\mathcal{I}_t[V_t \setminus \mathrm{bd}(V_t)]$. However, there might be several choices for the minimal vertex cover M of $G_{t,\bar{t}} - V(S)$. By the definition of the table entries, $M_t^{\mathrm{in}} \subseteq V_t \setminus V(\mathcal{I}_t)$ and M_t^{out} contains all vertices of $\overline{V_t}$ that have a neighbor in $V(\mathcal{I}_t) \setminus V(S)$. Throughout the remainder of the proof, we fix one such vertex cover M. Clearly, $\mathfrak{I}_t = ((S, M, Q), i, j)$ is a valid index according to the checks done in Step 0 and represents $\mathcal{I}_t$ in the sense of the definition of the table entries.

We observe that $\mathcal{I}_t$ induces partial solutions for the children a and b of t: We let $\mathcal{I}_a = \mathcal{I}_t \cap G[V_a \cup \mathrm{bd}(\overline{V_a})]$ and $\mathcal{I}_b = \mathcal{I}_t \cap G[V_b \cup \mathrm{bd}(\overline{V_b})]$. We now show how to obtain indices $\mathfrak{I}_a$ and $\mathfrak{I}_b$ representing $\mathcal{I}_a$ and $\mathcal{I}_b$, respectively, that are compatible to be combined to the index $\mathfrak{I}_t$ in the sense of Steps 1-4 of the algorithm presented above. In complete analogy to above, we obtain $S_a = \mathcal{I}_a \cap G_{a,\bar{a}}$, the pairing Q_a of degree one endpoints in $S_a[V_a]$, and the integers i_a and j_a and we proceed in the same way to obtain S_b, Q_b, i_b and j_b. Again, there will be several choices for the minimal vertex covers M_a of $G_{a,\bar{a}} - V(S_a)$ and M_b of $G_{b,\bar{b}} - V(S_b)$, of which we will choose one 'representative'. For the details see further below. What we can immediately verify is that S_a and S_b are compatible with S in the sense of Step 1, that Q_a and Q_b are compatible with Q in the sense of Step 2, that the values of j_a and j_b are compatible with j, and that i_a and i_b are compatible with i in accordance with Step 4. Throughout the following, we denote by $R = S_a \cap S_b$ and note that R is contained in the crossing graph $G_{a,b}$.

We now explain how to construct a pair of minimal vertex covers M_a and M_b of $G_{a,\bar{a}} - V(S_a)$ and $G_{b,\bar{b}} - V(S_b)$, respectively, making sure that they are compatible with M in the sense of Step 3 of the algorithm description. Note that in the following, we only show how to construct M_a and we perform the symmetric steps to construct M_b. Our strategy is as follows: We add vertices to M_a in consecutive stages and ensure in each stage that for each vertex x that we add to M_a,

- x covers an edge that has not been covered by M_a so far, and
- each neighbor of x has another neighbor that is not contained in M_a.

Hence minimality of the resulting vertex cover will follow. We furthermore point out at each stage, which part of Step 3 in the description of the join operation it satisfies.

1. Let u be an intermediate vertex of $\mathcal{I}_b$ and let x be a neighbor of u in V_a. Then, add x to M_a^{in}. Now, let u be an intermediate vertex of $\mathcal{I}_a$ and x be a neighbor of $\mathcal{I}_a$ in $\overline{V_a}$. Then, add x to M_a^{out}. In both cases, this covers the edge $\{u, x\}$. Since we do not add any more vertices to $M_a^{\mathrm{out}(b)}$ in the remaining construction, the vertex u in the first case will never be added to M_a and since in the second case, u is an intermediate vertex, it will not be added to M_a either. Hence this stage of the construction cannot violate the minimality condition of M_a. Note that since we proceed symmetrically for M_b, the condition in Step 3.1.1 is also satisfied by this part of the construction.

2. We add any $x \in M_t^{\text{in}} \cap V_a$ to M_a^{in}. Since M_t is minimal, x covers an edge $\{u, x\}$ in $G_{a,\bar{t}}$, where $u \in \overline{V_t} \setminus V(S_t)$. Hence, the edge $\{u, x\}$ has not been covered so far by M_a. This ensures that the condition in Step 3.1.2 a) is met.

3. If a vertex x in $V_a \setminus V(S_a)$ has a neighbor u in $V(S_b) \setminus V(S_a)$, then add x to M_a^{in}, so x covers the edge $\{u, x\}$. Note that this vertex u is different from the one in Stage 2, as $u \in V(S_b) \setminus V(S_a)$ and hence u is either a vertex of S_t or it is *not* contained in $\overline{V_t}$. This asserts that the condition in Step 3.2 is satisfied.

4. Consider the subgraph G^* of $G_{a,\bar{a}} - V(S_a)$ on the edges that have not been covered by M_a so far. By Stage 1, this graph does not touch any vertex in $\mathcal{I}_a$ or $\mathcal{I}_b$. We then add all vertices in $V(G^*) \cap V_a$ to M_a^{in}. This ensures that there is no vertex $v \in M_t^{\text{out}}$ that violates the condition of Step 3.1.2 b). Note that in this step, minimality of M_a is guaranteed as well, since all vertices in M_a^{out} have a neighbor in $\mathcal{I}_a$, and $M_a \cap V(\mathcal{I}_a) = \emptyset$.

By the above construction, in particular by Stage 4, we verify that M_a is a vertex cover of $G_{a,\bar{a}} - V(S_a)$ and in each stage, we checked that M_a remains minimal after adding the corresponding vertices.

We have shown how to derive from the partial solutions $\mathcal{I}_a$ and $\mathcal{I}_b$ a pair of table entries $\mathfrak{I}_a$, $\mathfrak{I}_b$ that represent them. Since we assume inductively that the algorithm is correct for the children of t, we know that $\mathcal{T}_a[\mathfrak{I}_a] = 1$ and $\mathcal{T}_b[\mathfrak{I}_b] = 1$. By the description of the algorithm, this implies that $\mathcal{T}_t[\mathfrak{I}_t] = 1$ which concludes this direction of the proof.

For the other direction, suppose there exists an index $\mathfrak{I}_t$ such that $\mathcal{T}_t[\mathfrak{I}_t] = 1$. By the description of the algorithm we can find pairs of indices $\mathfrak{I}_a$ and $\mathfrak{I}_b$ that are compatible with $\mathfrak{I}_t$ and such that $\mathcal{T}_a[\mathfrak{I}_a] = 1$ and $\mathcal{T}_b[\mathfrak{I}_b] = 1$. Assuming for the induction hypothesis that the values of the children of t are computed correctly, we obtain the corresponding partial solutions $\mathcal{I}_a$ and $\mathcal{I}_b$. Following the description of the algorithm, we can then see that $\mathcal{I}_a$ and $\mathcal{I}_b$ can be combined to a valid solution $\mathcal{I}_t$ which is represented by $\mathfrak{I}_t$. □

By Propositions 6 and 8 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 7, 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 9. *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 $(x_1, y_1), \ldots,$

(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 $G[V_t \cup \mathrm{bd}(\overline{V_t})]$, for some $t \in V(T)$. Recall that 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 $G[V_t \cup \mathrm{bd}(\overline{V_t})]$ 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 $G[V_t \cup \mathrm{bd}(\overline{V_t})]$. Note that each terminal $v_i \in \{x_i, y_i\}$ that is contained in $V_t \cup \mathrm{bd}(\overline{V_t})$ 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 ($\overline{V_t}$), yet $\mathcal{P}_i$ uses a vertex of $\overline{V_t}$ (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 $\mathcal{I}_{\lambda(C)}$. Remark that 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 $G[V_t \cup \mathrm{bd}(\overline{V_t})]$, 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 $v_i \in V(\mathcal{I}_i)$. 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.[5]

(a) $(s,t) \in Q_\lambda$ 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 $s,t \in V(\lambda^{-1}(i)) \cup \{x_i, y_i\}$.

(b) 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.

We now show that the answer to the problem can be obtained from inspecting the table entries stored in the root of T.

Proposition 10. *G contains a set of vertex-disjoint induced paths $\mathcal{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_k)$, where $\mathcal{P}_i$ is an (x_i, y_i)-path for each $i \in [k]$ and for $j \neq i$, no vertex in $\mathcal{P}_i$ is adjacent to a vertex in $\mathcal{P}_j$, if and only if $\mathcal{T}[r, (\emptyset, \emptyset, \emptyset, Q_\emptyset)] = 1$, where $Q_\emptyset = \{(x_1, y_1), \ldots, (x_k, y_k)\}$.*

Proof. Suppose that $\mathcal{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_k)$ is a solution to Induced Disjoint Paths, i.e. $\mathcal{P}$ satisfies the conditions of the proposition. First note that since $V_r = V(G)$ and $\mathrm{bd}(\overline{V_r}) = \emptyset$, $G[V_r \cup \mathrm{bd}(\overline{V_r})] = G$. This implies that $\mathcal{P}$ satisfies the conditions stated in Part (i) of the definition of a table entry being set to 1. (Clearly, each terminal $v_i \in \{x_i, y_i\}$ is contained in $V_r = V(G)$ and has degree one in $\mathcal{P}_i$.) Since $\mathrm{bd}(\overline{V_r}) = \emptyset$, it follows that $G_{r,\overline{r}} = \emptyset$ and hence, $S = \emptyset$, $M = \emptyset$ and $\lambda = \emptyset$. Since each $\mathcal{P}_i$ is an (induced) (x_i, y_i)-path and $Q_\emptyset$ pairs terminals (x_i, y_i) for each $i \in [k]$ by assumption, $Q_\emptyset$ satisfies Part (iv) of the definition of a table entry being set to 1. Remark that Part (iv.b) is satisfied since $\lambda = \emptyset$ and hence $\lambda^{-1}(i) = \emptyset$ for all $i \in [k]$. It follows that $\mathcal{T}[r, (\emptyset, \emptyset, \emptyset, Q_\emptyset)] = 1$.

Now suppose that $\mathcal{T}[r, (\emptyset, \emptyset, \emptyset, Q_\emptyset)] = 1$. Then by Part (i) of the definition of the table entries, there is an induced disjoint union of paths $\mathcal{I} = (\mathcal{I}_1, \ldots, \mathcal{I}_k)$ in $G[V_r \cup \mathrm{bd}(\overline{V_r})] = G$ such that for each $i \neq j$, no vertex in $\mathcal{I}_i$ is adjacent to a vertex in $\mathcal{I}_j$. Furthermore, for each $i \in [k]$, $\{x_i, y_i\} \subseteq V(\mathcal{I}_i)$, and both x_i and y_i have degree one in $\mathcal{I}_i$. Since $Q_\emptyset$ pairs all (x_i, y_i), Part (iv) allows us to conclude that each $\mathcal{I}_i$ is an (x_i, y_i)-path. □

Again, we denote by $\mathcal{S}_t$ the set of all induced unions of paths on at most $4w$ vertices in $G_{t,\bar{t}}$, for $S \in \mathcal{S}_t$ and by $\mathcal{M}_{t,S}$ the set of all minimal vertex covers of $G_{t,\bar{t}} - V(S)$. We let Λ_S denote the set of all labeling functions of the connected components of S and for $\lambda \in \Lambda_S$ by $\mathcal{Q}_{t,S,\lambda}$ the set of all pairings in accordance with Part (iv) of the table definition.

Proposition 11. *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 $n^{\mathcal{O}(w)}$.*

[5] 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 (i) and is incident to an edge in S, v_i cannot be paired with another vertex.

Proof. By the proof of Proposition 7, we know that $|\mathcal{S}_t| \leq n^{\mathcal{O}(w)}$ and $|\mathcal{M}_{t,S}| \leq \mathcal{O}(n^{4w})$. Since Lemma 4 also bounds the number of connected components in $S \in \mathcal{S}_t$ by $\mathcal{O}(w)$, we can conclude that $|\Lambda_S| \leq k^{\mathcal{O}(w)}$, since then for each $\lambda \in \Lambda_S$, $\lambda^{-1}(i) \neq \emptyset$ for at most $\mathcal{O}(w)$ values of $i \in [k]$. The same reasoning can be applied to count $|\mathcal{Q}_{t,S,\lambda}|$, since each $j \in [k]$ such that $\lambda^{-1}(j) = \emptyset$ allows no further choices for pairings in $\mathcal{Q}_{t,S,\lambda}$: Either both x_j and y_j are contained in V_t and we pair them, or neither of them is contained in V_t and we disregard them. We can conclude that $|\mathcal{Q}_{t,S,\lambda}| \leq w^{\mathcal{O}(w)}$. To summarize, there are at most

$$\mathcal{O}(n^{4w}) \cdot n^{\mathcal{O}(w)} \cdot k^{\mathcal{O}(w)} \cdot w^{\mathcal{O}(w)} = n^{\mathcal{O}(w)}$$

entries in $\mathcal{T}_t$ for each $t \in V(T)$. Note that even if $k = \mathcal{O}(n)$, we obtain a bound of $n^{\mathcal{O}(w)}$ on the number of table entries stored at each node $t \in V(T)$. The time bound on the enumeration of the indices follows from the same argument given in the proof of Proposition 7. □

We now explain how the table entries $\mathcal{T}_t$ are computed for each $t \in V(T)$.

Leaves of T. Let $t \in V(T)$ be a leaf of T and $v = \mathcal{L}^{-1}(t)$. Any nonempty intersection of an induced disjoint union of paths with $G_{t,\bar{t}}$ is either a single edge using v or a path on two edges having v as the middle vertex. Hence, all nonempty S such that the corresponding table entry is to 1 are either a single edge using v, and we denote this set by $\mathcal{S}^1_{t,v}$ or a path on two edges with v as the middle vertex and we denote this set by $\mathcal{S}^2_{t,v}$.

Suppose $S \in \mathcal{S}^1_{t,v}$. Since $G_{t,\bar{t}} - V(S)$ has no edges, $M = \emptyset$. If a solution intersects S, this means that v is a terminal vertex, i.e. $v \in \{x_{i^*}, y_{i^*}\}$ for some $i^* \in [k]$. Hence, (with a slight abuse of notation) $\lambda(S) = i^*$ and since v is the only degree one vertex in $S[V_t]$, $Q_\lambda = \emptyset$.

Now suppose $S \in \mathcal{S}^2_{t,v}$. First we observe that in this case, v cannot be a terminal vertex, i.e. $v \notin X$. Again, $G_{t,\bar{t}} - V(S)$ has no edges, so $M = \emptyset$. Let u_1, u_2 be the vertices in S other than v. We distinguish the following cases.

Case L1 (no terminal). If neither of u_1 and u_2 is a terminal vertex, then S can be a subgraph of any (x_i, y_i)-path, so we allow all choices of $\lambda(S) = i$ for $i \in [k]$.

Case L2 (one terminal). If precisely one of u_1 and u_2 is a terminal vertex for some $i^* \in [k]$, then the only choice for λ is such that $\lambda(S) = i^*$.

Case L3 (two terminals, same path). If $\{u_1, u_2\} = \{x_{i^*}, y_{i^*}\}$ for some $i^* \in [k]$, then the only choice for λ is such that $\lambda(S) = i^*$. Note that if $u_1 \in \{x_i, y_i\}$ and $u_2 \in \{x_j, y_j\}$ for $i \neq j$, then S is not a valid partial solution.

In all of the above cases, there is no degree one vertex in $S[V_t]$, so $Q_\lambda = \emptyset$.

Eventually, we have to consider the cases when no edge of $G_{t,\bar{t}}$ is used in a partial solution. Note that this is only possible if v is not a terminal vertex. The choices for the minimal vertex covers are then $\{v\}$ and $N(v)$ and since $S = \emptyset$, it follows that $\lambda = \emptyset$ and $Q_\lambda = \emptyset$.

To summarize, we set the table entries in $\mathcal{T}_t$ as follows.

$$\mathcal{T}[t,(S,M,\lambda,Q_\lambda)] = \begin{cases} 1, & \text{if } S \in \mathcal{S}^1_{t,v}, M = \emptyset, \lambda(S) = i^*, Q_\lambda = \emptyset \text{ and } v \in \{x_{i^*}, y_{i^*}\} \\ 1, & \text{if } S \in \mathcal{S}^2_{t,v}, M = \emptyset, \lambda(S) = i \text{ for some } i \in [k], Q_\lambda = \emptyset \\ & \text{and } \{v, u_1, u_2\} \cap X = \emptyset \text{ (Case L1)} \\ 1, & \text{if } S \in \mathcal{S}^2_{t,v}, M = \emptyset, \lambda(S) = i^*, Q_\lambda = \emptyset \\ & \text{and } u_{j_1} \in \{x_{i^*}, y_{i^*}\}, v \notin X, u_{j_2} \notin X \text{ where } \{j_1, j_2\} = [2] \text{ (Case L2)} \\ 1, & \text{if } S \in \mathcal{S}^2_{t,v}, M = \emptyset, \lambda(S) = i^*, Q_\lambda = \emptyset \\ & \text{and } \{u_1, u_2\} = \{x_{i^*}, y_{i^*}\}, v \notin X \text{ (Case L3)} \\ 1, & \text{if } S = \emptyset, M \in \{\{v\}, N(v)\}, \lambda = \emptyset, Q_\lambda = \emptyset \text{ and } v \notin X \\ 0, & \text{otherwise} \end{cases}$$

Internal nodes of T. We argue that for each internal node $t \in V(T)$ and each index $\mathfrak{I}_t := (S, M, \lambda, Q_\lambda) \in \mathcal{S}_t \times \mathcal{M}_{t,S} \times \Lambda_S \times \mathcal{Q}_{t,S,\lambda}$, the value of the table entry $\mathcal{T}[\mathfrak{I}_t]$ can be computed in complete analogy with the algorithm for Longest Induced Path (LIP). In particular, it can be viewed as performing for each $i \in [k]$ the steps presented in the algorithm of LIP. Some attention must be devoted to guaranteeing that there is no edge between two components labeled i and j (where $i \neq j$). However, as outlined below, partial solutions to the two problems represented by their respective indices have almost exactly the same structure and in that sense, the routine presented for LIP already captures this requirement. More precisely, the only way in which unwanted edges could be added during the join in Induced Disjoint Paths (IDP) is if they cross the boundary. By definition, there will never be any unwanted edges in the intersection of a partial solution with the boundary in both problems. Subsequently, the only place where they can appear is between intermediate vertices of a combined partial solution. In the join of LIP, we explicitly ensure that no edges between intermediate vertices appear (via the minimal vertex cover) and in the same way we can enforce in IDP that no edges appear between intermediate vertices of components labeled i and j (for $i \neq j$) when combining two partial solutions.

We now argue in detail the similarity between the partial solution based on the definitions of the table entries for the respective problems. Throughout the following, we refer to the conditions stated in the definition of the table entries for Longest Induced Path by LIP($\cdot$) and for Induced Disjoint Paths by IDP($\cdot$).

First, observe that in both cases, the intersection of a solution with the graph $G[V_t \cup \mathrm{bd}(\overline{V_t})]$ is an induced disjoint union of paths $\mathcal{I}$. The key difference between the two problems is that in LIP we are interested in the solution size, whereas in IDP we are not and that in IDP we additionally have to take into account to which (x_i, y_i)-path the components of $\mathcal{I}$ belong. Note that aside from the size constraint, LIP(i) and IDP(i) express precisely the same thing if $k = 1$ in IDP and the entry j in LIP takes the role of the fixed terminals in IDP.

LIP(ii) and IDP(ii.a) are in fact identical, in particular this means that the intersection S of a partial solution with the crossing graph $G_{t,\bar{t}}$ is the same in both problems. So to make the join procedure described for LIP work for IDP in a way that it preserves Condition IDP(ii), we only have to take into account the behavior of the labeling function λ introduced in IDP(ii.b). Note that again, if $k = 1$ in IDP, then LIP(ii) and IDP(ii) express precisely the same thing. LIP(iii) and IDP(iii) are as well identical, so all parts of the algorithm for LIP that ensure that this condition holds can immediately applied (and argued for) in the join of IDP.

It remains to argue the similarity of LIP(iv) and IDP(iv). Since we do not know the endpoints of the induced path we are looking for in LIP, whereas in IDP we do, slightly differing languages

were used to express the properties of the pairings. In LIP(iv), two vertices are paired if contracting all edges in $\mathcal{I} - E(G_{t,\bar{t}})$ leaves an edge between the two vertices in the graph. But this happens precisely when there is a path between these two vertices in $\mathcal{I}[V_t]$, the condition stated in IDP(iv.a). The remainder of IDP(iv) is devoted to including the terminals in the pairings, so the necessary modifications in the algorithm of LIP to work for IDP such that it respects IDP(iv) are straightforward. Note that again if $k = 1$ in IDP, LIP(iv) and IDP(iv) express the same thing where j takes the role of the terminals contained in V_t.

By the above discussion, the fact that we enumerate all possible solutions in the leaves of T, Propositions 10 and 11 and in the light of the correctness (and runtime) of the join of the algorithm for Longest Induced Path, we have the following theorem.

Theorem 12. *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.

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

Theorem 13. *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 $(T, \mathcal{L})$.*

Proof. Let H be a fixed graph. To solve H-Induced Topological Minor, we have to find a map φ from the vertices of H to a set of vertices in G such that there is an edge in $\{x, y\}$ if and only if there is an induced path between $\varphi(x)$ and $\varphi(y)$ in G such that additionally, for each pair P_1, P_2 of such paths, no vertex in P_1 is adjacent to a vertex in P_2. We do so by guessing to which vertices in G the vertices of H are mapped and after some preprocessing, we run the algorithm for Induced Disjoint Paths with (at most) $|E(H)|$ pairs of terminals to find the induced paths in G corresponding to the edges in H (see Figure 4).

Step 1 (Branching). For each vertex $x \in V(H)$, we guess the corresponding vertex $v_x \in V(G)$ of x, i.e. $\varphi(x) = v_x$. Hence we require that for all $x, y \in V(H)$, $x = y \Leftrightarrow v_x = v_y$ and that $\deg_G(v_x) \geq \deg_H(x)$. We let $X := \bigcup_{x \in V(H)} v_x$. Furthermore, for each $x \in V(H)$ and neighbor y of x, we guess a distinct corresponding neighbor $v_{x,y}$ in G. Note that since $\deg_G(v_x) \geq \deg_H(x)$ for all $x \in V(H)$, there are sufficiently many such vertices in G to choose from. The vertex $v_{x,y}$ is the vertex adjacent to v_x in the induced path in G corresponding to the edge $\{x, y\}$ in H. We let $Y_x := \bigcup_{y \in N_H(x)} v_{x,y}$ and $Y := \bigcup_{x \in V(H)} Y_x$ and we refer to the vertices in Y as the *neighbor vertices*. We branch on each such choice of $X \cup Y$.

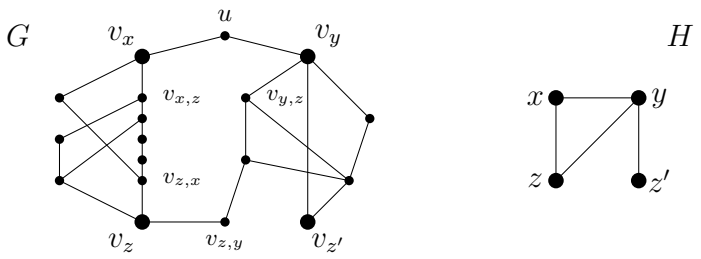

Figure 4: An example YES-instance of H-INDUCED TOPOLOGICAL MINOR. Note that $v_{y,z'} = v_{z'}$ and $v_{z',y} = v_y$ since $\{v_y, v_{z'}\} \in E(G)$ in accordance with Step 2(a). For each of the remaining edges in H there are precisely two corresponding edges each in G, illustrating Step 2(b). Furthermore, $u = v_{x,y} = v_{y,x}$, illustrating the special case dealt with in Step 2(c).

Step 2 (Preprocessing according to $X \cup Y$). In this step we check whether the current choice of X and Y is valid and prepare the current instance for the call to the algorithm of INDUCED DISJOINT PATHS.

(a) For each edge $\{x, y\} \in E(H)$, if $\{v_x, v_y\} \in E(G)$, then we remove the edge $\{x, y\}$ from H without changing the answer to the problem: $\{v_x, v_y\}$ is the induced path in G corresponding to the edge $\{x, y\}$. If on the other hand, $\{x, y\} \notin E(H)$ but $\{v_x, v_y\} \in E(G)$, then we discard this choice of $X \cup Y$.

(b) We check whether for each edge $\{x, y\} \in E(H)$ there are precisely two corresponding edges, namely $\{v_x, v_{x,y}\}$ and $\{v_y, v_{y,x}\}$ in G and that $G[X \cup Y]$ induces no further edges. In particular, if $G[X \cup Y]$ does contain any additional edges, we discard this choice of X and Y.

(c) If there is a vertex $u \in V(G)$ which is chosen more than once as a neighbor vertex, i.e. there exists a set $A \subseteq V(H)$ such that $u \in \bigcap_{a \in A} Y_a$ with $|A| > 1$, then we proceed only if $A = \{x, y\}$ for some $x, y \in V(H)$ and such that $u = v_{x,y} = v_{y,x}$. In that case, we remove the edge $\{x, y\}$ from H without changing the answer to the problem: The set $\{v_x, u, v_y\}$ induces the path in G corresponding to the edge $\{x, y\}$. Furthermore, we remove the vertices in $N(u) \setminus X$ from G, since these vertices cannot be used by any other path corresponding to an edge in H.

Step 3 (Execution of IDP-algorithm). We run the algorithm for INDUCED DISJOINT PATHS on $G - X$ with pairs of terminals $(v_{x,y}, v_{y,x})_{\{x,y\} \in E(H)}$. (Note that some edges of H might have been removed in Steps 2(a) and 2(c) and that some vertices of G might have been removed in Step 2(c).)

Step 4 (Return). We let the algorithm return YES if at least one run of INDUCED DISJOINT PATHS in Step 3 returned YES, and NO otherwise.

We now analyze the runtime of the algorithm. In Step 1, we branch in $n^{\mathcal{O}(|E(H)|)}$ ways, the number of choices for the sets X and Y. The checks in Step 2 can be performed in time polynomial in n and each execution of the algorithm for INDUCED DISJOINT PATHS in Step 3 takes time $n^{\mathcal{O}(w)}$ by Theorem 12. So the total runtime of the algorithm is $n^{\mathcal{O}(|E(H)|)} \cdot n^{\mathcal{O}(w)} = n^{\mathcal{O}(w)}$, as H is fixed. □

4 Hamiltonian Cycle for linear mim-width 1

In this section we contrast the algorithmic results of the previous section with a proof that the Hamiltonian Cycle problem remains NP-complete on graphs of linear mim-width 1. We recall the problem definition.

Hamiltonian Cycle
Input: A graph G
Question: Is there a cycle $C \subseteq G$ such that $V(C) = V(G)$?

The next theorem shows that the above problem parameterized by linear mim-width is para-NP-complete.

Theorem 14. Hamiltonian Cycle *is* NP*-complete on graphs of linear mim-width* 1*, and the hardness holds even if a linear branch decomposition of mim-width* 1 *is given.*

Proof. Itai et al [14] showed that given a bipartite graph G with maximum degree 3, it is NP-complete to decide if it has a Hamiltonian cycle, while Panda and Pradhan [26] construct, from this graph G, a rooted directed path graph H such that H has a Hamiltonian cycle if and only if G does. Here we show that the construction of [26] can be used to also output a linear mim-width 1 decomposition of H, in polynomial time, which will prove our result.

Let G be the bipartite graph on bipartition $(\{v_1, v_2, \ldots, v_m\}, \{w_1, w_2, \ldots, w_m\})$ that has maximum degree at most 3, and has no leaves. Let us consider the construction, given by Panda and Pradhan [26], of a new graph H from G as follows:

1. For each $i \in \{1, \ldots, m\}$, we introduce vertices X_i, Y_i to H, and if w_i has degree 3, then we introduce a vertex Z_i additionally. We let $\mathcal{X}_{\geq i} := \bigcup_{i' \geq i} X_{i'}$, $\mathcal{X} := \mathcal{X}_{\geq 1}$, $\mathcal{Y}_{\geq i} := \bigcup_{i' \geq i} Y_{i'}$, $\mathcal{Y} := \mathcal{Y}_{\geq 1}$, $\mathcal{Z}_{\geq i} := \bigcup_{\substack{i' \geq i \\ \deg(w_i)=3}} Z_{i'}$ and $\mathcal{Z} := \mathcal{Z}_{\geq 1}$.

2. For each $v_i w_j \in E(G)$, we introduce a vertex $A_{i,j}$ to H. We let $\mathcal{A}_j := \bigcup_{i:\, v_i w_j \in E(G)} A_{i,j}$ and $\mathcal{A} := \bigcup_{j \leq m} \mathcal{A}_j$.

3. For each $i \in \{1, \ldots, m\}$, we make $\{X_i\} \cup \{A_{i',j} : i' \leq i, v_{i'} w_j \in E(G)\}$ a clique in H.

4. For each $j \in \{1, \ldots, m\}$, we make $\{Y_j\} \cup \{A_{i,j} : v_i w_j \in E(G)\}$ a clique in H, and w_i has degree 3, then we also make $\{Z_j\} \cup \{A_{i,j} : v_i w_j \in E(G)\}$ a clique in H.

That concludes the construction of the graph H. We now summarize the most important properties of H which will be useful later in the proof.

(H1) $\mathcal{X} \cup \mathcal{Y} \cup \mathcal{Z}$ is an independent set in H.

(H2) $\mathcal{A}$ is a clique in H.

(H3) For $j \in [m]$ and $v \in \{Y_j, Z_j\}$, $N(v) = \mathcal{A}_j$.

(H4) For $i_1, i_2 \in [m]$ with $i_1 \leq i_2$, $N(X_{i_1}) \subseteq N(X_{i_2})$ and for all $j \in [m]$, $N(A_{i_1,j}) \cap \mathcal{X} \subseteq N(A_{i_2,j}) \cap \mathcal{X}$.

(H5) For $j \in [m-1]$, no vertex in $\mathcal{A}_j$ has a neighbor in $\mathcal{Y}_{\geq j+1} \cup \mathcal{Z}_{\geq j+1}$.

Formally, a branch decomposition $(T, \mathcal{L})$ of H is linear if T consists of a path on $|V(H)| - 2$ nodes with a leaf added to each inner node of the path, with $\mathcal{L}$ a bijection between the leaves of T and the vertices of H. For simplicity, let us say that a linear branch decomposition is a total ordering of the vertices of H. For each $j \in \{1, \ldots, m\}$, let L_j be any ordering of vertices in $\mathcal{A}_j$, and let U_j be the ordering (Y_j, Z_j) if w_j has degree 3, and (Y_j) otherwise. We claim that the linear branch decomposition[6]

$$U_1 \oplus L_1 \oplus U_2 \oplus L_2 \oplus \cdots \oplus U_m \oplus L_m \oplus (X_m, X_{m-1}, \ldots, X_1) \tag{1}$$

of H has mim-width at most 1. Let $v \in V(H)$, and let S_v be the union of $\{v\}$ and the set of vertices appearing before v in the ordering, and let $T_v := V(H) \setminus S_v$. We divide into three cases depending on the place where the vertex v appears in the linear branch decomposition. Suppose for a contradiction that there exist $a_1, a_2 \in S_v$, $b_1, b_2 \in T_v$ such that a_1b_1, $a_2b_2 \in E(H)$ but $a_1b_2, a_2b_1 \notin E(H)$ (which would imply that the linear branch decomposition (1) has mim-width at least two).

Case 1 ($v \in U_k$ for some k). By (H3), every vertex in U_t where $t < k$, has no neighbors in T_v. Let $x \in T_v$ be a neighbor of v. By (H1), $x \in \mathcal{A}$ so by (H2), x is adjacent to every vertex in $L_1 \cup L_2 \cup \cdots \cup L_k$. We can conclude that v is neither a_1 nor a_2. We have argued that $a_1, a_2 \in L_1 \cup \cdots \cup L_{k-1}$. By (H4), either $N(a_1) \cap \mathcal{X} \subseteq N(a_2) \cap \mathcal{X}$ or $N(a_2) \cap \mathcal{X} \subseteq N(a_1) \cap \mathcal{X}$. Suppose wlog. that the former holds. Together with (H2) and (H5), we can conclude that $N(a_1) \cap T_v \subseteq N(a_2) \cap T_v$, a contradiction.

Case 2 ($v \in L_k$ for some k). Again by (H3), every vertex in U_t where $t < k$ has no neighbors in T_v. By the argument given in Case 1, it cannot happen that a_1 and a_2 are both contained in $L_1 \cup \cdots \cup L_{k-1} \cup (L_k \cap S_v)$. Hence we can (wlog.) assume that $a_1 \in U_k$, i.e. $a_1 = Y_k$ or $a_1 = Z_k$. Since Y_k and Z_k are twins by (H3), a_2 cannot be the vertex in $U_k \setminus \{a_1\}$ (if exists). We can conclude that $a_2 \in L_1 \cup \cdots \cup L_{k-1} \cup (L_k \cap S_v)$, in particular that $a_2 \in \mathcal{A}$. By (H3), $N(a_1) = \mathcal{A}_k$, so by (H2), every neighbor of a_1 is adjacent to a_2, a contradiction.

Case 3 ($v = X_k$ for some k). Suppose wlog. that $b_1 = X_{j_1}$ and $b_2 = X_{j_2}$ where $j_1 < j_2 < k$. By (H4), $N(b_1) \subset N(b_2)$, so in particular $N(b_1) \cap S_v \subset N(b_2) \cap S_v$, a contradiction.

We have shown that the linear branch decomposition (1) has mim-width 1. □

5 Conclusion and Outlook

In this first volume in a series of papers regarding algorithmic properties of the graph parameter mim-width, we have initialized the study of the complexity, parameterized by mim-width, of problems that are not locally checkable. In particular, we have shown that the Longest Induced Path, Induced Disjoint Paths and H-Induced Topological Minor problems are solvable in time $n^{\mathcal{O}(w)}$ when the input graph is accompanied by one of its branch decompositions of mim-width at most w. In contrast, we showed that Hamiltonian Cycle remains NP-complete on graphs of linear mim-width 1.

The latter result shows that the class of graphs of linear mim-width 1 is larger than previously known; so far on all known classes of linear mim-width 1, Hamiltonian Cycle is polynomial-time

[6]For two disjoint total orderings X and Y we define their sum $X \oplus Y$ as follows. Suppose $X = (x_1, x_2, \ldots, x_r)$ and $Y = (y_1, y_2, \ldots, y_s)$, then $X \oplus Y = (x_1, x_2, \ldots, x_r, y_1, y_2, \ldots, y_s)$.

solvable. This shift in the understanding of the mim-width parameter deems it worthwhile to further investigate graph classes that have linear mim-width 1 and their structural properties.

The big open question revolving around the parameter mim-width (see e.g. [28, 30]) is whether one can compute in XP time an $f(w)$-approximation to mim-width w. A polynomial-time algorithm that given a graph either outputs one of its branch decompositions of mim-width at most c (where c is a fixed constant), or concludes that the (linear) mim-width of a graph is greater than 1 would be an important step towards tackling this difficult question.

We now give an outlook on the volumes of the series that are in preparation. In Volume II [18], we provide an $n^{\mathcal{O}(w)}$ time algorithm for the Feedback Vertex Set problem where w, the parameter, denotes the mim-width of a given branch decomposition of the input graph; and show that the problem is W[1]-hard in this parameterization. In Volume III [19], we prove bounds on the mim-width of powers of graphs of bounded tree-width, clique-width, and mim-width. The latter result implies that a plethora of generalizations of distance domination problems is solvable in time $n^{\mathcal{O}(w)}$ when given a mim-width-w branch decomposition of the input graph. We also show that many classes of generalized domination problems are W[1]-hard parameterized by mim-width.

We observe that all parameterized problems that admit polynomial-time algorithms for graphs of bounded mim-width are in XP and many of them are being shown to be W[1]-hard (e.g. [18, 19], see also [15]). This gives rise to the question of whether *any* natural graph problem parameterized by mim-width can be shown to be solvable in FPT time or whether there is a unified (formal) explanation of why the existence of such FPT algorithms is implausible.

Acknowledgements. We thank an anonymous reviewer for many valuable suggestions that improved the quality of the presentation in this paper.

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.

[2] Rémy Belmonte and Martin Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theoret. Comput. Sci.*, 511:54–65, 2013.

[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] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[5] Jitender S. Deogun and George Steiner. Polynomial algorithms for hamiltonian cycle in cocomparability graphs. *SIAM Journal on Computing*, 23(3):520–552, 1994.

[6] Reinhard Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Heidelberg, fourth edition, 2010.

[7] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[8] 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.

[9] Fanica Gavril. Algorithms for maximum weight induced paths. *Inform. Process. Lett.*, 81(4):203 – 208, 2002.

[10] Petr A. Golovach, Daniël Paulusma, and Erik Jan van Leeuwen. Induced disjoint paths in at-free graphs. In *SWAT 2012*, pages 153–164, 2012.

[11] 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.

[12] Petr Hliněný, Sang-il Oum, Detlef Seese, and Georg Gottlob. Width parameters beyond tree-width and their applications. *The Computer Journal*, 51(3):326–362, 2008.

[13] Tetsuya Ishizeki, Yota Otachi, and Koichi Yamazaki. An improved algorithm for the longest induced path problem on k-chordal graphs. *Discrete Appl. Math.*, 156(15):3057 – 3059, 2008.

[14] Alon Itai, Christos H. Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM J. Comput.*, 11(4):676–686, 1982.

[15] Lars Jaffke, O-joung Kwon, and Jan Arne Telle. A note on the complexity of feedback vertex set parameterized by mim-width, 2017. arXiv:1711.05157 [cs.CC].

[16] 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. In *IPEC 2017*, pages 21:1–21:13, 2017.

[17] Lars Jaffke, O-joung Kwon, and Jan Arne Telle. A unified polynomial-time algorithm for feedback vertex set on graphs of bounded mim-width. In *STACS 2018*, pages 42:1–42:14, 2018.

[18] Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Mim-width II. The feedback vertex set problem. To appear in Algorithmica, 2019.

[19] Lars Jaffke, O-joung Kwon, Torstein J. F. Strømme, and Jan Arne Telle. Mim-width III. Graph powers and generalized distance domination problems. submitted, 2019.

[20] Dong Yeap Kang, O-joung Kwon, Torstein J.F. Strømme, and Jan Arne Telle. A width parameter useful for chordal and co-comparability graphs. *Theoretical Computer Science*, 704:1–17, 2017.

[21] Ken-ichi Kawarabayashi and Yusuke Kobayashi. *The Induced Disjoint Paths Problem*, pages 47–61. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[22] J. Mark Keil. Finding hamiltonian circuits in interval graphs. *Information Processing Letters*, 20(4):201–206, 1985.

[23] Dieter Kratsch, Haiko Müller, and Ioan Todinca. Feedback vertex set and longest induced path on at-free graphs. In *WG 2003*, pages 309–321, 2003.

[24] Haiko Müller. Hamiltonian circuits in chordal bipartite graphs. *Discrete Mathematics*, 156(1-3):291–298, 1996.

[25] Sridhar Natarajan and Alan P. Sprague. Disjoint paths in circular arc graphs. *Nordic J. of Computing*, 3(3):256–270, September 1996.

[26] B. S. Panda and D. Pradhan. NP-completeness of hamiltonian cycle problem on rooted directed path graphs. *CoRR*, abs/0809.2443, 2008.

[27] Neil Robertson and Paul D. Seymour. Graph minors. XIII. The disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

[28] 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.

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

[30] Martin Vatshelle. *New Width Parameters of Graphs*. PhD thesis, University of Bergen, 2012.