An FPT Algorithm for Spanning Trees with Few Branch Vertices Parameterized by Modular-Width

Luisa Gargano
Department of Computer Science, University of Salerno, Italy

Adele A. Rescigno
Department of Computer Science, University of Salerno, Italy

Abstract

The minimum branch vertices spanning tree problem consists in finding a spanning tree T of an input graph G having the minimum number of branch vertices, that is, vertices of degree at least three in T. This NP-hard problem has been widely studied in the literature and has many important applications in network design and optimization. Algorithmic and combinatorial aspects of the problem have been extensively studied and its fixed parameter tractability has been recently considered. In this paper we focus on modular-width and show that the problem of finding a spanning tree with the minimum number of branch vertices is FPT with respect to this parameter.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Mathematics of computing → Graph theory

Keywords and phrases Spanning Trees, Branch vertices, Fixed-parameter tractable algorithms, Modular-width

Digital Object Identifier 10.4230/LIPIcs.MFCS.2023.50

Funding This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU.

1 Introduction

Let $G = (V, E)$ be an undirected graph where V is the set of vertices and E is the set of edges. Given a spanning tree T of G, a *branch vertex* is a vertex having degree at least three in T. If G is a connected graph, we denote by $b(G)$ the smallest number of branch vertices in any spanning tree of G. We study the following problem:

Minimum Branch Vertices (MBV)
Instance: A connected graph $G = (V, E)$.
Goal: Find a spanning tree of G having $b(G)$ branch vertices.

Notice that a spanning tree of G without branch vertices is a Hamilton path, that is, $b(G) = 0$ if and only if G admits a Hamilton path.

The problem of determining a spanning tree with a bounded number of branch vertices, while a natural theoretical question, was introduced to solve a problem related to wavelength-division multiplexing technology in optical networks, where one wants to minimize the number of light-splitting switches in a light-tree [11]. Also for Cognitive Radio Networks other than for 5G technologies, that operate with a wide range of frequencies, bounding the switching costs due to the switching between different service providers has high importance both in terms of delay and energy consumption [16, 24]. MBV has been then widely studied, both from the algorithmic and the graph-theoretic point of view. Gargano *et al.* [12] proved that it is NP-complete to decide whether a graph G admits a spanning tree with at most k branch vertices, for given G and k, even in cubic graphs. Salamon [23] proved the existence of an algorithm that finds a spanning tree with $O(\log |V(G)|)$ branch vertices whenever the

degree of each vertex of the input graph is $\Omega(n)$; moreover, an approximation factor better than $O(\log |V(G)|)$ would imply that $NP \subseteq DTIME(n^{O(\log \log n)})$. Sufficient conditions for a connected claw-free graph to have a spanning tree with k branch vertices are given in [20]. Integer linear formulations of MBV and some variants are presented in [5–7], together with different relaxations of them; the authors also provide numerical result comparison of the considered relaxations. In [26] hybrid integer linear programs for MB are considered and solved with branch-and-cut algorithms. In [18, 21, 22] decomposition methods of graphs are used to solve the MBV problem. Other heuristics are presented in [19, 25, 27]. In [8] a complementary formulation of MBV is investigated. It is called maximum path-node spanning tree (MPN), where the goal is to find a spanning tree that maximizes the number of vertices with degree at most two; the authors prove that MPN is APX-hard and present an approximation algorithm with ratio 6/11. Related gathering processes are considered in [2–4, 13, 14].

1.1 Parameterized Complexity

Parameterized complexity is a refinement to classical complexity theory in which one takes into account not only the input size, but also other aspects of the problem given by a parameter p. A problem with input size n and parameter p is called *fixed parameter tractable (FPT)* if it can be solved in time $f(p) \cdot n^c$, where f is a computable function only depending on p and c is a constant.

In this paper we are interested in assessing the complexity of MBV when parameterized by modular-width. It was recently proven that MBV is FPT when parameterized either by treewidth [1] or by neighborhood diversity [15]. On the other hand, it was shown in [9] that the problem is $W[1]$-hard when parameterized by clique-width. Specifically, in [9] it was proven that the (MBV special case) hamiltonian path problem is $W[1]$-hard when parameterized by clique-width. See Figure 1 for a relation among the above parameters.

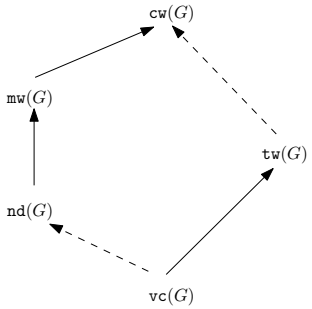

Figure 1 A summary of the relations holding among some popular parameters. We use mw(G), tw(G), cw(G), nd(G), and vc(G) to denote modular-width, treewidth, cliquewidth, neighborhood diversity, and minimum vertex cover of a graph G, respectively. Solid arrows denote generalization, e.g., modular-width generalizes neighborhood diversity. Dashed arrows denote that the generalization may exponentially increase the parameter.

1.2 Modular-width

Modular-width was introduced in [10] as graph parameter which could cover dense graphs but still allows FPT algorithms for the problems lost to clique-width.

▶ **Definition 1** (Modular-width [10]). *Consider graphs that can be obtained from an algebraic expression that uses the following operations:*

- *(O1) Create an isolated vertex;*
- *(O2) the disjoint union of 2 graphs denoted by $G_1 \oplus G_2$, i.e., $G_1 \oplus G_2$ is the graph with vertex set $V(G_1) \cup V(G_2)$ and edge set $E(G_1) \cup E(G_2)$;*
- *(O3) the complete join of 2 graphs denoted by $G_1 \otimes G_2$, i.e., $G_1 \otimes G_2$ is the graph with vertex set $V(G_1) \cup V(G_2)$ and edge set $E(G_1) \cup E(G_2) \cup \{\{v, w\} : v \in V(G_1)$ and $w \in V(G_2)\}$;*
- *(O4) the substitution operation with respect to some graph G with vertex set $\{1, 2, \ldots, n\}$ i.e., for graphs $G_1, \ldots, G_n$ the substitution of the vertices of G by the graphs $G_1, \ldots, G_n$, denoted by $G(G_1, \ldots, G_n)$, is the graph with vertex set $\bigcup_{i=1}^{n} V(G_i)$ and edge set $\bigcup_{i=1}^{n} E(G_i) \cup \{\{u, v\} \mid u \in V(G_i), v \in V(G_j), \{i, j\} \in E(G)\}$. Hence, $G(G_1, \ldots, G_n)$ is obtained from G by substituting every vertex $i \in V(G)$ with the graph G_i and adding all edges between the vertices of a graph G_i and the vertices of a graph G_j whenever $\{i, j\} \in E(G)$.*

Let A be an algebraic expression that uses only the operations (O1)–(O4). The width of A is the maximum number of operands used by any occurrence of the operation (O4) in A. The modular-width of a graph H, denoted $\mathtt{mw}(H)$, is the least integer m such that H can be obtained from such an algebraic expression of width at most m.

We recall that an algebraic expression of width $\mathtt{mw}(G)$ can be constructed in linear time [28].

Given a graph $H = G(G_1, \ldots, G_n)$, we will refer to the graphs $G_1, \ldots, G_n$ also as the modules of H. Notice that given the graph $H = G(G_1, \ldots, G_n)$, by the operations O(1)-(O4), one has that all the vertices of G_i share the same neighborhood outside G_i; indeed,

$$\begin{array}{ll} \{\{u, v\} \mid u \in V(G_i), v \in V(G_j)\} \subseteq E(H) & \text{if } \{i, j\} \in E(G) \\ \{\{u, v\} \mid u \in V(G_i), v \in V(G_j)\} \cap E(H) = \emptyset & \text{if } \{i, j\} \notin E(G) \end{array} \tag{1}$$

for each $i, j = 1, \ldots, n$ with $i \neq j$.

1.3 Graph Partitioning

A *spider* is a tree with at most one branch vertex. The *center* of the spider is the branch vertex, if it exists, and is any vertex otherwise. A *path-spider cover of a graph G* is a set composed by one spider and some paths that are pair-wise (vertex-)disjoint and whose union contains every vertex of G. We denote by $spi(G)$ the least integer p such that G has a path-spider cover with $p - 1$ paths.

In order to solve MBV, we define and study the following problem that can be of its own interest:

Path-Spider Cover (PSC)
Instance: A graph $G = (V, E)$.
Goal: Find a path-spider cover of G with $spi(G) - 1$ paths.

Moreover, we will need the following Partitioning into Paths problem that was proven to be FPT with respect to modular-width in [10]. A *partition of a graph G into paths* is a set of (vertex-)disjoint paths of G whose union contains every vertex of G. We denote by $ham(G)$ the least integer p such that G has a partition into p paths. Notice that $spi(G) \leq ham(G)$.

Partitioning into Paths (PP)
Instance: A graph $G = (V, E)$.
Goal: Find a partition of G into $ham(G)$ paths.

As originally defined in [10], the Partitioning into Paths problem only asks for the value $ham(G)$, while we ask for the actual path partitioning of G.

In the following we will denote by $\mathcal{P}_{ham(G)}$ a partition of G into $ham(G)$ paths, and by $\mathcal{P}_{spi(G)}$ a path-spider cover of G with $spi(G) - 1$ paths.

Given a path P in G, we will denote by $f(P)$ and $s(P)$ the two end-points of P; we will distinguish them as *the first and the second end-point of* P, respectively. Furthermore, if P denotes a spider in G then we will equally use either $f(P)$ or $s(P)$ to denote *the center of* P .

2 Our Results

We present an FPT algorithm for MBV parameterized by modular-width. To this aim, we also design a FPT algorithm for PSC parameterized by modular-width.

Let H be the input graph. Consider the parse-tree of an algebraic expression describing H, according to the rules (O1)-(O4) in Section 1.3. We take a look at the operation corresponding to the root: Operation (O1) is trivial and (O2) yields a disconnected graph, therefore we suppose the last operation is either (O3) or (O4). Hence, we can see the input graph as $H = G(G_1, \ldots, G_n)$ where G is a graph with $n \leq \texttt{mw}(H)$ vertices and $G_1, \ldots, G_n$ are graphs.

The algorithm that finds a spanning tree of an input graph H with $b(H)$ branch vertices goes through the following steps 1) and 2).

1) An FPT algorithm for PSC and PP parameterized by the modular-width of the input graph H. Namely, for each vertex $\hat{H} = \hat{G}(\hat{G}_1, \ldots, \hat{G}_{\hat{n}})$ of the parse-tree of H, we show how to compute the triple

$$(ham(\hat{H}), spi(\hat{H}), |V(\hat{H})|) \text{ together with } \mathcal{P}_{ham(\hat{H})} \text{ and } \mathcal{P}_{spi(\hat{H})},$$

where $\mathcal{P}_{ham(\hat{H})}$ is a partition of $\hat{H}$ into $ham(\hat{H})$ paths and $\mathcal{P}_{spi(\hat{H})}$ is a path-spider cover of $\hat{H}$ with $spi(\hat{H}) - 1$ paths.

2) Compute a spanning tree of $H = G(G_1, \ldots, G_n)$ with $b(H)$ branch vertices by using the values, computed at step 1), for the graphs $G_1, \ldots, G_n$, that is,

$$(ham(G_i), spi(G_i), |V(G_i)|), \ \mathcal{P}_{ham(G_i)}, \text{ and } \mathcal{P}_{spi(G_i)},$$

for $i = 1, \ldots, n$,

The computation in step 2) is only done once, i.e., for the root vertex of the parse tree, corresponding to the input graph $H = G(G_1, \ldots, G_n)$. It is shown in Section 3, which is devoted to prove the following theorem.

▶ **Theorem 2.** Minimum Branch Vertices *parameterized by modular-width is fixed-parameter tractable.*

The computation in step 1) is presented in Section 4. Following [10], we use a bottom-up dynamic programming approach along the parse-tree to compute for every vertex a record of data, using those already computed for its children. Since the operations of type (O1)-(O3) can be replaced by one operation of type (O4) that uses at most 2 operands, we only focus on the computation (and the time it requires) of a record of data for a vertex of type (O4) in the parse-tree. Namely, in Section 4 we prove the following theorem.

▶ **Theorem 3.** Path-Spider Cover *parameterized by modular-width is fixed-parameter tractable.*

3 The MBV algorithm

In this section we give an algorithm that finds a spanning tree of graph H with $b(H)$ branch vertices. We assume here that for each vertex $\hat{H} = \hat{G}(\hat{G}_1, \ldots, \hat{G}_n)$ of the parse-tree of H, we already have

the triple $(ham(\hat{H}), spi(\hat{H}), |V(\hat{H})|)$ together with $\mathcal{P}_{ham(\hat{H})}$ and $\mathcal{P}_{spi(\hat{H})}$.

We start by giving a characterization of a spanning tree with the minimum number of branch vertices in terms of the modular decomposition of H.

▶ **Lemma 4.** *Let $H = G(G_1, \ldots, G_n)$ be a connected graph. There exists a spanning tree of H with $b(H)$ branch vertices that has at most one branch vertex belonging to G_i for each $i = 1, \ldots, n$. Hence, $b(H) \leq n \leq \mathtt{mw}(H)$.*

Proof. Let T be a spanning tree of H with $b(H)$ branch vertices. Denote by B the set of vertices of H that are branch vertices in T and by $N_T(v)$ the set of neighbors of v in T, for any vertex v. Assume that $|V(G_i) \cap B| \geq 2$ for some $i \in \{1, \ldots, n\}$. We show how to transform T so to satisfy the lemma. The transformation consists of two phases.

- **Phase 1.** For each $i = 1, \ldots, n$, we denote by B_i the set of branch vertices in $V(G_i)$ that have in T at least two neighbors outside $V(G_i)$, that is,

  $$B_i = \{v \mid v \in V(G_i) \cap B \text{ and } |N_T(v) \cap (\cup_{j \neq i} V(G_j))| \geq 2\}.$$

  In this phase we transform T so that $|B_i| \leq 1$, for each i. We proceed as follows.

  For each i such that $|B_i| \geq 2$,
  - let v be any node in B_i;
  - for each $w \in B_i$ with $w \neq v$, consider the path connecting v and w in T, say $v, \ldots, w', w$, and modify T as follows: For any $x \in (N_T(w) - V(G_i)) - \{w'\}$, substitute in T the edge $\{w, x\}$ by the edge $\{v, x\}$.
    (Notice that this is possible by (1) and implies $B_i = \{v\}$).

- **Phase 2.** We know that each G_i contains at most one branch vertex with at least two neighbors outside $V(G_i)$, that is now $|B_i| \leq 1$ for each i.
  If there exists i such that G_i contains at least 2 branch vertices, we modify the spanning tree so that only one remains. We proceed as follows.

  While there exists i such that $|V(G_i) \cap B| \geq 2$.
  - Choose any $j \neq i$ such that $\{i, j\} \in E(G)$ and let

    $$w \in \begin{cases} B_j & \text{if } B_j = \{w\}, \\ V(G_j) \cap B & \text{if } B_j = \emptyset \text{ and } V(G_j) \cap B \neq \emptyset \\ V(G_j) & \text{otherwise.} \end{cases}$$

  - For each branch vertex $v \in V(G_i) \cap B$ with $v \neq w$, perform the following step.
    * Consider the path connecting v and w in T, say $v, v', \ldots w$, and modify T as follows: For any $x \in N_T(v) \cap V(G_i)$ and $x \neq v'$, substitute in T the edge $\{v, x\}$ by the edge $\{w, x\}$.
      (This is possible by (1). Moreover, even if now w becomes a new branch vertex, we know that $|V(G_j) \cap B| = 1$; finally, $|B_j| \leq 1$, $|V(G_i) \cap B| \leq 1$), and the number of branch vertices does not increase.

  By iterating the above steps, one can obtain the desired spanning tree of H with at most one branch vertex in each $V(G_i)$. ◀

In the remaining part of this section, we present an algorithm that computes a spanning tree of $H = G(G_1, \ldots, G_n)$ with $b(H)$ branch vertices, if $b(H) > 0$. In Section 4.2 we deal with the case $b(H) = 0$, that is, we show how to find a Hamiltonian path of H, if any exists.

By exploiting Lemma 4, the algorithm proceeds by considering all the subsets $B_G \subseteq \{1, \ldots, n\}$ with $|B_G| \geq 1$, ordered by size, and checking whether there exists a spanning tree of H with $|B_G|$ branch vertices, so that exactly one branch vertex belongs to each $V(G_i)$ with $i \in B_G$ and none to each $V(G_i)$ with $i \notin B_G$.

The identification of the spanning tree of H goes through the solution of an Integer Linear Program that uses the values $ham(G_i), spi(G_i), |V(G_i)|$, for $i = 1, \ldots, n$, and exploits property (1). Namely, if the ILP does not admit a solution for B_G, then the set is discarded; if for B_G the ILP admits a solution, we will show how to use the partition of G_i given in $\mathcal{P}_{ham(G_i)}$ and $\mathcal{P}_{spi(G_i)}$ to construct a spanning tree of H having exactly $|B_G|$ branch vertices (recall that the sets B_G are considered by increasing size). The optimal spanning tree will be indeed shown to correspond to the smallest B_G for which the ILP admits a solution.

3.1 The Integer Linear Program

Let $B_G \subseteq \{1, \ldots, n\}$, with $|B_G| \geq 1$. Construct a digraph

$$G_{B_G} = (\{1, \ldots, n\} \cup \{s\}, A_{B_G}),$$

where $s \notin V(H)$ is an additional vertex that will be called the source. G_{B_G} is obtained from G by replacing each edge $\{i, j\} \in E(G)$ by the two directed arcs (i, j) and (j, i), and then adding a directed arc (s, r) where r is an arbitrary vertex in B_G. Formally,

$$A_{B_G} = \{(s, r)\} \cup \{(i, j), (j, i) \mid \text{there exists an edge between } i \text{ and } j \text{ in } E(G)\}.$$

For sake of clearness, we will refer to the vertices of G as *module indices* and reserve the term vertex to those in H.

We use the solution of the following Integer Linear Programming (ILP) to select arcs of G_{B_G} that will help to construct the desired spanning tree in H.

$$x_{sr} = 1 \tag{2}$$

$$\sum_{j:(j,i)\in A_{B_G}} x_{ji} \leq |V(G_i)| \qquad \forall i \in \{1, \ldots, n\} \tag{3}$$

$$\sum_{j:(j,i)\in A_{B_G}} x_{ji} \geq spi(G_i) \qquad \forall i \in B_G \tag{4}$$

$$\sum_{j:(j,i)\in A_{B_G}} x_{ji} \geq ham(G_i) \qquad \forall i \in \{1, \ldots, n\} - B_G \tag{5}$$

$$\sum_{\ell:(i,\ell)\in A_{B_G}} x_{i\ell} - \sum_{j:(j,i)\in A_{B_G}} x_{ji} \leq 0 \qquad \forall i \in \{1, \ldots, n\} - B_G \tag{6}$$

$$y_{sr} = n \tag{7}$$

$$\sum_{j:(j,i)\in A_{B_G}} y_{ji} - \sum_{\ell:(i,\ell)\in A_{B_G}} y_{i\ell} = 1 \qquad \forall i \in \{1, \ldots, n\} \tag{8}$$

$$y_{ij} \leq n\, x_{ij} \qquad \forall (i, j) \in A_{B_G} \tag{9}$$

$$y_{ij}, x_{ij} \in \mathbb{N} \qquad \forall (i, j) \in A_{B_G} \tag{10}$$

For each arc $(i,j) \in A_{B_G}$, the non negative decision variable x_{ij} represents the load to be put on (i,j). The load of the arc (s,r) is set to 1. The total incoming load at module index $i \in \{1,\ldots,n\}$ has to be at most $|V(G_i)|$ and at least $spi(G_i)$ in case $i \in B_G$ (to be sure that the spider and all the $spi(G_i)-1$ paths in G_i are reached) and at least $ham(G_i)$ in case $i \notin B_G$ (to be sure that all the $ham(G_i)$ paths in G_i are reached). Constraints (3), (4) and (5) correspond to this requirement.
Constraint (6) binds the relation between the total incoming and outgoing loads at any $i \notin B_G$, namely i must have an outgoing load upper bounded by its incoming load.
Constraints (7) and (8) use a single commodity flow in which s is used as the source and the other module indices are demand vertices. For each arc $(i,j) \in A_{B_G}$, the non negative decision variable y_{ij} represents the quantity of flow from i to j.
Each $i \in \{1,\ldots,n\}$ has demand of one unit; therefore, the difference between the inflow and the outflow must be exactly one. Meanwhile, the supply quantity at the source s has to be exactly n, in order to reach each of the module index in $\{1,\ldots,n\}$.
Constraint (9) stresses variable $y_{ij} = 0$ whenever $x_{ij} = 0$; thus if no load is put on (i,j) then j cannot be reached trough i.

Given an integer solution (y,x), if any, to the above ILP, the values of variables y imply that each module index $i \in \{1,\ldots,n\} - \{r\}$ is reached from the source s. Then, by the construction of the digraph G_{B_G}, each module index $i \in \{1,\ldots,n\}$ is reached from module index r. Furthermore, by the relation between variables x and y (constraint (9)), we know that each module index $i \in \{1,\ldots,n\}$ gets incoming load from at least one of its neighbors.

▷ Claim 5. The subgraph G_x of G_{B_G} with vertex set $\{1,\ldots,n\}$ and arc set $\{(i,j)|\ x_{ij} \geq 1\}$ contains a directed path from r to any other module index .

We stress that the constraints involving variables y only assure that a spanning tree in G_x exists. A more sophisticate approach is necessary to find a spanning tree of H with branch vertices in B_G only.

3.2 The spanning tree construction

Our algorithm constructs a spanning tree T of H with $|B_G|$ branch vertices, one in each $V(G_i)$ with $i \in B_G$. To this aim, it uses the values of variables x and the path-spider cover $\mathcal{P}_{spi(G_i)}$ for $i \in B_G$ and the partition into disjoint paths $\mathcal{P}_{ham(G_i)}$ for $i \notin B_G$.

Denote by $In(i)$ the set of the module indices for which there exist arcs in G_x toward i, that is, $In(i) = \{j\ |x_{ji} \geq 1\}$, and by

$$\alpha_i = \sum_{j:j\in In(i)} x_{ji} \tag{11}$$

the number of vertices of $V(G_i)$ whose parent in T is a vertex outside $V(G_i)$.
Let $\mathcal{P}^i = \{P^i_1, P^i_2, \ldots, P^i_{\alpha_i}\}$ be

- the path-spider cover of G_i obtained from those in $\mathcal{P}_{spi(G_i)}$ by removing $\alpha_i - spi(G_i)$ arbitrary edges in case $i \in B_G$ (notice that by constraint (4), it holds $\alpha_i \geq spi(G_i)$), or
- the partition of G_i into disjoint paths obtained from those in $\mathcal{P}_{spi(G_i)}$ by removing $\alpha_i - ham(G_i)$ arbitrary edges in case $i \notin B_G$ (notice that by (4), it holds $\alpha_i \geq ham(G_i)$).

Furthermore, denote by

$$f(\mathcal{P}^i) = \{f(P^i_1), f(P^i_2), \ldots, f(P^i_{\alpha_i})\}$$

the sets of the first end-points in the partition $\mathcal{P}^i$ and by

$$s(\mathcal{P}^i) = \{s(P^i_1), s(P^i_2), \ldots, s(P^i_{\alpha_i})\}$$

the sets of the second end-points in $\mathcal{P}^i$. In case $i \in B_G$, we assume that $P_1^i \in \mathcal{P}^i$ is the spider and $f(P_1^i) = s(P_1^i)$ is the center in P_1^i.
We also denote by

$$\beta_i = \begin{cases} \sum_{\ell : i \in In(\ell)} x_{i\ell} & \text{if } i \notin B_G \\ 1 & \text{if } i \in B_G \end{cases} \tag{12}$$

the number of vertices of $V(G_i)$, that will be the parent of some vertex in $\bigcup_{\ell : i \in In(\ell)} V(G_\ell)$.

Our algorithm ensures that the α_i vertices in $f(\mathcal{P}^i)$ are the vertices in G_i whose parent in T is outside $V(G_i)$, and that β_i vertices among those in $s(\mathcal{P}^i)$ are chosen to be parents of vertices outside $V(G_i)$. Notice that by Claim 5 ($\alpha_i \geq 1$) and constraint (6), it follows that $\alpha_i \geq \beta_i$ for each $i \in \{1, \ldots, n\}$.

Figure 2 shows the partition of graph G_i (whose vertices are grouped in the dotted circle) into α_i disjoint paths if $i \notin B_G$ and into a spider plus $\alpha_i - 1$ disjoint paths if $i \in B_G$.

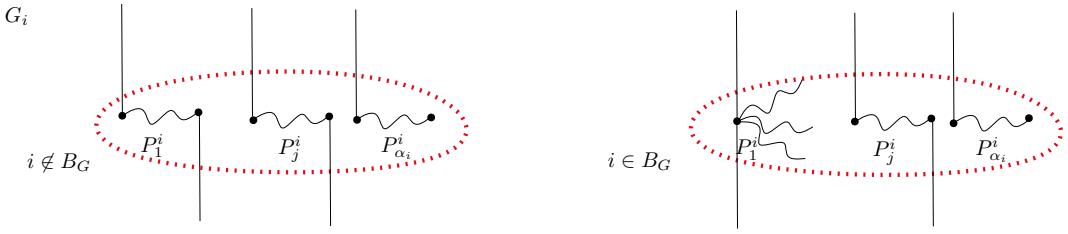

Figure 2 The vertices of graph G_i, grouped in the dotted circle, as partitioned in α_i disjoint paths if $i \notin B_G$ and in a spider plus $\alpha_i - 1$ disjoint paths if $i \in B_G$. Vertex $f(P_j^i)$ is the only vertex in P_j^i whose parent in T is outside G_i and vertex $s(P_j^i)$ is the only vertex in P_j^i that can have a children in T outside G_i.

The algorithm TREE constructs a spanning tree of $H = G(G_1, \ldots, G_n)$ iteratively by exploring unexplored vertices of H, until possible, and maintains a main subtree T and a forest whose roots are progressively connected to T to assemble the spanning tree. The process stops when all the vertices of H are explored. A similar idea was used in [15] in the special case in which each graph G_i is either a clique or an independent set.

The algorithm uses a queue Q to enqueue the explored vertices and maintains a set R of the roots of trees of explored vertices that wait to be connected to the main tree T. The forest structure is described through the parent function π.

At the beginning the set R is empty. The exploration starts with the center $f(P_1^r)$ of the spider in the path-spider cover of G_r (recall that by construction $r \in B_G$); the procedure EXPLORE($f(P_1^r)$) carries out the construction of the main tree T rooted at $f(P_1^r)$ and marks as explored all the reached vertices (adding them to the set Ex). Clearly, for each explored vertex v there is a path in T joining $f(P_1^r)$ to v.

However, it can occur that some of the vertices have not been explored (i.e., $V(H) - Ex \neq \emptyset$). In such a case an explored vertex $w \in f(\mathcal{P}^j) \cap Ex$ is chosen so that it belongs to some $V(G_j)$ which also contains at least a unexplored vertex $u \in (f(\mathcal{P}^j) - Ex) - R$ which is able to explore at least one unexplored neighbour outside G_j, that is, $\beta_j \geq 1$ (the existence of such a set $V(G_j)$ is assured by Lemma 7). By using (1) and knowing that the parents of vertices in $f(\mathcal{P}^j)$ are outside $V(G_j)$, the algorithm makes:

- the parent of w (recall that w is explored) become the parent of u, and
- w (the root of a subtree of explored vertices) is added to R and removed from Ex (this allows to later explore w and add it, together with its subtree, to the main tree T), and
- EXPLORE(u) is called to start a new exploration from u.

Notice that the algorithm modifies the forest by assigning to u the parent of w and only later (after adding u and some descendants of u) adding again the subtree rooted in w to the main tree T. This allows connecting new vertices in $V(H) - Ex$ to the main tree T; the particular choice of u and w will be shown to avoid the possibility that the algorithm fails, due to the fact that no arc can be added to T without forming a cycle or creating an extra branch vertex. The process is iterated as long as there are unexplored vertices, i.e. $V(H) - Ex \neq \emptyset$.

The procedure EXPLORE(u) starts exploring $u = f(P_k^j)$ together with the whole P_k^j *, then putting in Q only the vertex $s(P_k^j)$ and successively padding the main tree T (recall that (unless $u = f(P_1^r)$) the parent of u is a vertex already in T, thus we construct a subtree rooted at u spanning on all the newly explored vertices). EXPLORE(u) uses for each module index i the values of α_i and β_i that are initially defined as in (11) and (12), and the partition $\mathcal{P}^i = \{P_1^i, P_2^i, \ldots, P_{\alpha_i}^i\}$ of G_i. The value $\alpha_i = \sum_{j:j\in In(i)} x_{ji}$ counts the number of vertices of $V(G_i)$ that must be assigned a parent outside $V(G_i)$, they are the vertices in $f(\mathcal{P}^i) = \{f(P_1^i), f(P_2^i), \ldots, f(P_{\alpha_i}^i)\}$; in particular, x_{ji} vertices of $f(\mathcal{P}^i)$ have to be explored by vertices in $V(G_j)$, for $j \in In(i)$. The value β_i counts the number of vertices of $V(G_i)$ that have to explore other vertices in some other $V(G_\ell)$, for $\ell : i \in In(\ell)$; in particular,

- if $i \in B_G$ then exactly $\beta_i = 1$ vertex in $V(G_i)$, that is $s(P_1^i)$ (i.e., the center of the spider P_1^i), becomes a branch vertex in T: it is set as the parent of $x_{i\ell}$ unexplored vertices in $f(\mathcal{P}^\ell)$ for each ℓ such that $x_{i\ell} \geq 1$ (i.e., $i \in In(\ell)$), and
- if $i \notin B_G$ then $\beta_i = \sum_{\ell:i\in In(\ell)} x_{i\ell}$ vertices in $s(\mathcal{P}^i) = \{s(P_1^i), s(P_2^i), \ldots, s(P_{\alpha_i}^i)\}$ are chosen and each one becomes the parent of one unexplored vertex in $f(\mathcal{P}^\ell)$.

  Recall that, by the ILP constraints, we know that $\alpha_i \geq \beta_i$.

The vertices in $s(\mathcal{P}^i)$ for $i \in \{1, \ldots, n\}$ are the only one to be enqueued in Q. When a vertex $v \in s(\mathcal{P}^i)$ is dequeued from Q in EXPLORE(u) then the value of β_i is decreased by one if v explores (i.e., if $\beta_i \geq 1$). In this case, for each explored vertex $f(P_h^\ell)$, with $i \in In(\ell)$, the whole P_h^ℓ is also explored. Furthermore, the value α_ℓ is decreased by the number of vertices in $s(\mathcal{P}^\ell)$ that v explores, for $i \in In(\ell)$. Hence, at the beginning of each iteration of the while loop in EXPLORE(u) the values of α_i represents the number of vertices in $f(\mathcal{P}^i)$ that remain to be explored while β_i is the number of vertices in $s(\mathcal{P}^i)$ that already have to explore. Note that when a vertex $v \in s(\mathcal{P}^i)$ is dequeued from Q in EXPLORE(u), with $u \neq f(P_1^r)$, and $\beta_i \geq 1$, the algorithm checks if the neighbour $v' \in f(\mathcal{P}^\ell)$, that v explores, is in R (i.e., v' is a root of a tree in the forest). In this case v', with the tree rooted at it, is connected to the main tree T (since it was already explored in the past).

▶ **Lemma 6.** *At the end of EXPLORE($f(P_1^r)$) the function π describes a tree, rooted at $f(P_1^r)$, spanning the set $Ex \subseteq V(H)$ of explored vertices. The vertices in $B \cap Ex$ are the branch vertices.*

Proof. When EXPLORE($f(P_1^r)$) is called, the whole spider P_1^r is explored (i.e. $Ex = Ex \cup P_1^r$ and so added to T) and its center $s(P_1^r)$ is enqueued in Q. After that, each time a vertex $v \in s(\mathcal{P}^i)$ is dequeued from Q (recall, $v \in Ex$, i.e., it is an explored vertex), the algorithm can either stop its exploration (i.e., $\beta_i = 0$) or explore one o more unexplored neighbor of v together with the path/spider it belongs. Indeed, we can prove that v has the needed number of unexplored neighbors. If $\beta_i = 0$ then v is a leaf in T; hence, we only have to consider the case $\beta_i \geq 1$. If $i \notin B_G$ then v has $\beta_i \geq 1$ unexplored neighbors and one of them, say $f(P_h^\ell)$

* Assume that when in the algorithm $P \in \mathcal{P}^j$ is explored, that is $Ex = Ex \cup P$, then the parent function π is set going through all the vertices in P from $f(P)$ to $s(P)$ in case P is a path, and from the center $f(P) = s(P)$ to the leaves in case P is a spider.

Algorithm 1 TREE$(H, G_1, \ldots, G_n, r, B_G)$.

1: $R = \emptyset, \quad B = \emptyset, \quad Ex = \emptyset$
2: $\pi(u) = nhil \quad$ for each $u \in V(H)$
3: EXPLORE$(f(P_1^r))$
4: **while** $V(H) - Ex \neq \emptyset$ **do**
5: - Let G_j be any graph s.t. $((f(\mathcal{P}^j) - Ex) - R \neq \emptyset \neq f(\mathcal{P}^j) \cap Ex)$ and $\beta_j \geq 1$
6: - Let $w \in f(\mathcal{P}^j) \cap Ex$ and $u \in (f(\mathcal{P}^j) - Ex) - R$
7: - Set $\pi(u) = \pi(w), \quad Ex = Ex - \{w\}, \quad R = R \cup \{w\}$
8: - EXPLORE(u)
9: **end while**
10: **return** $\pi, \ B$

for $i \in In(\ell)$, can be added to T as child of v . If $i \in B_G$ then v is the first vertex of $V(G_i)$ that explores and $x_{i\ell}$ vertices in $f(\mathcal{P}_\ell)$ are unexplored and can be added to T as children of v, for each ℓ such that $x_{i\ell} \geq 1$. Hence v becomes a branch vertex in T and is put in B. Since $R = \emptyset$ (i.e, no tree is in the forest), each time a neighbor of v is explored, say $f(P_h^\ell)$, then the whole path/spider P_h^ℓ is added to T and $s(P_h^\ell)$ is enqueued in Q. Hence, any explored vertex has $f(P_1^r)$ has ancestor, i.e., the function π describes a path joining any explored vertex to $f(P_1^r)$. Noticing that no vertex can be enqueued twice in Q (since any enqueued vertex is also marked as explored), we have that the function π does not create cycles. ◀

We can also prove the following result.

▶ **Lemma 7.** *Let Ex be the set of explored vertices at the beginning of any iteration of the while loop in algorithm TREE. If $V(H) - Ex \neq \emptyset$ then there exists a module index j such that $(f(\mathcal{P}^j) - Ex) - R \neq \emptyset \neq f(\mathcal{P}^j) \cap Ex$ and $\beta_j \geq 1$.*

▶ **Lemma 8.** *After each call of EXPLORE(u) the function π describes a forest spanning the vertices in $Ex \cup R$ of explored vertices and consisting of $|R| + 1$ trees respectively rooted at $f(P_1^r)$ and at the vertices in R. The vertices in B are the only branch vertices in the forest.*

Proof. When EXPLORE(u) is called, the function π describes a forest, spanning the current set $Ex \cup R$, whose roots are the vertices in $\{f(P_1^r)\} \cup R$ and where $R \subset V(H) - Ex$. We notice that by Lemma 6, this is true the first time EXPLORE is called, that is, after the call to EXPLORE$(f(P_1^r))$ (at that time $R = \emptyset$).

We prove that the claim is also true at the end of each call to EXPLORE(u). When EXPLORE(u) is called, Q is empty; vertex u is explored (i.e. it is added to Ex) and enqueued in Q. Then EXPLORE(u) proceeds, exactly as in EXPLORE$(f(P_1^r))$, dequeueing vertices from Q and exploring their unexplored neighbors, so constructing a subtree of the main tree T rooted at u described by function π. The only difference with EXPLORE$(f(P_1^r))$ is when one of the vertices explored is $v' \in R$. Vertex $v' \in R$ is removed from R (see lines 11, 22) and connected to the main tree T through the function π and marked as explored exactly as any other explored vertex. However v' is not enqueued in Q since it has already explored its neighbors; hence, v' is connected to T together with its subtree of explored vertices. ◀

We are now able to prove the following result.

▶ **Lemma 9.** *The algorithm TREE returns a spanning tree of H, described by function π, with branch vertex set B.*

Proof. By using Lemma 6 we know that algorithm TREE constructs, through procedure EXPLORE$(f(P_1^r))$, a main tree T, described by π. In case T does not span all the vertices in $V(H)$ then, Lemma 7 assures that the algorithm finds a graph G_j with an explored vertex

Algorithm 2 EXPLORE(u).

```
1: Let Q be an empty queue
2: Let u = f(P_k^j)
3: Ex = Ex ∪ P_k^j
4: Q.enqueue(s(P_k^j))
5: while Q ≠ ∅ do
6:     v = Q.dequeue
7:     Let v ∈ s(P^i)
8:     if i ∉ B_G and β_i ≥ 1 then
9:         - Let f(P_h^ℓ) ∈ f(P^ℓ) − Ex for some ℓ s.t. i ∈ In(ℓ)
10:        - π(f(P_h^ℓ)) = v
11:        if f(P_h^ℓ) ∉ R then
12:            - Ex = Ex ∪ P_h^ℓ
13:            - Q.enqueue(s(P_h^ℓ))
14:        else R = R − {f(P_h^ℓ)}
15:        end if
16:        - α_ℓ = α_ℓ − 1
17:        - β_i = β_i − 1
18:    else if i ∈ B_G and β_i = 1 then
19:        - B = B ∪ {v}
20:        for each ℓ s.t. i ∈ In(ℓ) do
21:            - Let A_iℓ ⊆ f(P^ℓ) − Ex s.t. |A_iℓ| = x_iℓ
22:            - α_ℓ = α_ℓ − x_iℓ,
23:            for each f(P_h^ℓ) ∈ A_iℓ do
24:                - π(f(P_h^ℓ)) = v
25:                if f(P_h^ℓ) ∉ R then
26:                    -Ex = Ex ∪ P_h^ℓ
27:                    -Q.enqueue(s(P_h^ℓ))
28:                else R = R − {f(P_h^ℓ)}
29:                end if
30:            end for
31:        end for
32:        - β_i = β_i − 1
33:    end if
34: end while
```

$w \in f(\mathcal{P}^j) \cap Ex$ and an unexplored vertex $u \in (f(\mathcal{P}^j) - Ex) - R$. Disconnecting w (together with its subtree) from the main tree T, the algorithm let w become one of the roots of trees in R. Furthermore, since the parent of w in T is a vertex outside $V(G_j)$ and, since u and w share the same neighborhood outside G_j (by (1)), the algorithm let u be connected to the vertex that was the parent of w in T (thus, connecting u to T). Considering that $u \notin R$ and $\beta_j \geq 1$, the algorithm starts a new exploration from u (recall that $u \in f(\mathcal{P}^j) - Ex$) calling EXPLORE($u$). By Lemma 8, this allows padding T with the subtree rooted a u. The lemma follows by iterating the above procedure until no unexplored vertex exists in $V(H)$. ◀

3.3 The algorithm complexity

Summarizing, given the triple $(ham(G_i), spi(G_i), |V(G_i)|)$ together with $\mathcal{P}_{ham(G_i)}$ and $\mathcal{P}_{spi(G_i)}$, for each $i \in \{1, \ldots, n\}$, the proposed method to construct the spanning tree of $H = G(G_1, \ldots, G_n)$ works as follows.
For each $B_G \subseteq \{1, \ldots, n\}$ with $|B_G| \geq 1$, selected in order of increasing size, the algorithm executes the following steps:

- solve the corresponding ILP
- if a solution exists for the current set B_G, use algorithm TREE to construct a spanning tree of $H = G(G_1, \ldots, G_n)$ with $|B_G|$ branch vertices.

Jansen and Rohwedderb [17] have recently showed that the time needed to find a feasible solution of an ILP with p integer variables and q constraints is $O(\sqrt{q}\Delta)^{(1+o(1))q} + O(qp)$, where Δ is the largest absolute value of any coefficient in the ILP. Denoted by m the number of edges of G, our ILP has $q = 3n + 2m + 1$ constraints, $p = 2(m + 1)$ variables and $\Delta = n$. Hence the time to solve it is $O(n\sqrt{n+m})^{(1+o(1))(3n+2m+1)} + O(n(n+m))$. Using the solution (y, x) of the ILP, the algorithm TREE returns the spanning tree of H in time $O(|V(H)|^2)$. Overall, the algorithm requires time

$$2^n[O(n\sqrt{n+m})^{(1+o(1))(3n+2m+1)} + O(n(n+m))] + O(|V(H)|^2).$$

Recall that $n \leq \mathtt{mw}$, and therefore $m \leq \mathtt{mw}^2$.

3.4 Optimality

It is possible to show that if no set $B_G \subseteq \{1, \ldots, n\}$, of size k exists for which the ILP admits a solution then any spanning tree of $H = G(G_1, \ldots, G_n)$ has $b(H) \geq k + 1$ branch vertices. This allows to say that the optimal spanning tree in H corresponds to the smallest set $B_G \subseteq \{1, \ldots, n\}$ for which the ILP admits a solution, if any. Namely, we can prove the following result.

▶ **Lemma 10.** *Given the graphs $G_1, \ldots, G_n$ and $ham(G_i), spi(G_i), |V(G_i)|$ for each $i = 1, \ldots, n$, if there exists a spanning tree in $H = G(G_1, \ldots, G_n)$ with $k \geq 1$ branch vertices then there exist a set $B_G \subseteq \{1, \ldots, n\}$ with $|B_G| = k$, for which ILP admits a solution (x, y).*

4 The triple and partition computation

In this section we show how to compute the record of data for any vertex $\hat{H} = \hat{G}(\hat{G}_1, \ldots, \hat{G}_{\hat{n}})$ of the parse-tree. Namely, given the triple $(ham(\hat{G}_i), spi(\hat{G}_i), |V(\hat{G}_i)|)$, $\mathcal{P}_{ham(\hat{G}_i)}$ and $\mathcal{P}_{spi(\hat{G}_i)}$, for each $i = 1, \ldots, \hat{n}$, we have to compute

the triple $(ham(\hat{H}), spi(\hat{H}), |V(\hat{H})|)$ together with $\mathcal{P}_{ham(\hat{H})}$ and $\mathcal{P}_{spi(\hat{H})}$,

Clearly, $|V(\hat{H})| = \sum_{i=1}^{\hat{n}} |V(\hat{G}_i)|$. We show below how to compute $spi(\hat{H})$ and $\mathcal{P}_{spi(\hat{H})}$, and also $ham(\hat{H})$ and $\mathcal{P}_{ham(\hat{H})}$,

For a graph $\hat{H}$ and an integer ℓ, we denote by $\hat{H} \otimes \ell$ the graph obtained from $\hat{H}$ by adding ℓ vertices and connecting them to every vertex in $\hat{H}$; formally, $\hat{H} \otimes \ell$ has vertex set $V(\hat{H}) \cup \{v_1, \ldots, v_\ell\}$ and edge set $E(\hat{H}) \cup \{\{u, v_j\} \mid u \in V(\hat{H}), 1 \leq j \leq \ell\}$. We notice that, since $\hat{H} = \hat{G}(\hat{G}_1, \ldots, \hat{G}_{\hat{n}})$ then the graph $\hat{H} \otimes \ell$, for each $2 \leq \ell \leq |V(\hat{H})|$, is equal to the graph $\hat{G}'(\hat{G}_1, \ldots, \hat{G}_{\hat{n}}, I_\ell)$ where $\hat{G}'$ is the graph obtained from $\hat{G}$ by adding the vertex $\hat{n} + 1$ (i.e., $V(\hat{G}') = \{1, \ldots, \hat{n}, \hat{n} + 1\}$) and making it adjacent to all the other vertices of $\hat{G}$ (i.e., $E(\hat{G}') = \{(i, \hat{n} + 1) \mid 1 \leq i \leq \hat{n}\}$), and I_ℓ is the independent set with ℓ vertices $\{v_1, \ldots, v_\ell\}$.

4.1 Computing $spi(\hat{H})$ and $\mathcal{P}_{spi(\hat{H})}$

In order to compute the values $spi(\hat{H})$ and $\mathcal{P}_{spi(\hat{H})}$, we first need a preliminary result.

▶ **Proposition 11.** *Let $\hat{H}$ be a graph and*

$$s(\hat{H}) = \min\{\ell \mid \hat{H} \otimes (\ell - 1) \text{ has a spanning spider with center in } \hat{H}\}.$$

Then $spi(\hat{H}) = s(\hat{H})$.

Proof. We first show that $s(\hat{H}) \leq spi(\hat{H})$. Let $P_1, P_2, \ldots, P_{spi(\hat{H})}$ be the path-spider cover of $\hat{H}$ and let $f(P_1)$ be the center of spider P_1 and, $f(P_i)$ be the first end-point of path P_i for $i = 2, \ldots, spi(\hat{H})$. Hence, the graph $\hat{H} \otimes (spi(\hat{H}) - 1)$ contains the spider with center $f(P_1)$ obtained connecting $f(P_1)$ to the vertex i and then connecting vertex i to $f(P_i)$ for each for $i = 1, \ldots, spi(\hat{H}) - 1$. Now, we prove that $spi(\hat{H}) \leq s(\hat{H})$. Let S be a spider in $\hat{H} \otimes (s(\hat{H}) - 1)$ with the center $u \in V(\hat{H})$. Then, removing from S the vertices in $\{1, \ldots, s(\hat{H}) - 1\}$ we have a path-spider cover with center u and $s(\hat{H}) - 1$ path pairwise disjoint. ◀

Recall that the graph $\hat{H} \otimes (\ell - 1)$ is equal to $\hat{G}'(\hat{G}_1, \ldots, \hat{G}_n, I_{\ell-1})$ and notice that

$$ham(I_{\ell-1}) = spi(I_{\ell-1}) = \ell - 1 \text{ and } \mathcal{P}_{ham(I_{\ell-1})} = \mathcal{P}_{spi(I_{\ell-1})} = I_{\ell-1}.$$

We can then take into account the values $ham(\hat{G}_i)$, $spi(\hat{G}_i)$, $|V(\hat{G}_i)|$, and the sets $\mathcal{P}_{ham(\hat{G}_i)}$ and $\mathcal{P}_{spi(\hat{G}_i)}$, for all $i \in \{1, \ldots, \hat{n}\}$. For each $B_{\hat{G}'} = \{j\}$, for $j = 1, \ldots, \hat{n}$, we can follow the lines of Section 3.1 to verify whether the corresponding ILP is feasible. In the positive case, following the construction given in Section 3.2, one can obtain a spider of $\hat{H} \otimes (\ell - 1)$ centered in $V(\hat{G}_i)$

The minimum ℓ for which the above occurs, gives $spi(\hat{H})$ as well as the spider T covering $\hat{H} \otimes (spi(\hat{H}) - 1)$ with center in $V(\hat{H})$. The arguments used in Section 2.3 allows to obtain the time complexity of this computation (here, $\hat{m}$ represents the number of edges of $\hat{G}$)

$$spi(\hat{H})\ \hat{n}\ [O(\hat{n}\sqrt{\hat{n} + \hat{m}})^{(1+o(1))(3\hat{n}+2\hat{m}+1)} + O(\hat{n}(\hat{n} + \hat{m}))] + O(|V(\hat{H})|^2).$$

Clearly, the subgraph of T induced by $V(\hat{H})$ returns the path-spider cover $\mathcal{P}_{spi(\hat{H})}$ of $\hat{H}$, thus concluding the proof of Theorem 3.

4.2 Computing $ham(\hat{H})$ and $\mathcal{P}_{ham(\hat{H})}$

Using an approach similar to the one in the proof of Proposition 11, we can prove the following result.

▶ **Proposition 12.** *Let $\hat{H}$ be a graph and*

$$h(\hat{H}) = \min\{\ell \mid \hat{H} \otimes \ell \text{ has a hamiltonian path with an end-point in } \{v_1, \ldots, v_\ell\}\}.$$

Then $ham(\hat{H}) = h(\hat{H})$.

By Proposition 12, the value $ham(\hat{H})$ is equal to the minimum positive integer ℓ with $1 \leq \ell \leq |V(\hat{H})|$ such that the graph

$$\hat{H} \otimes \ell \text{ has a hamiltonian path with an end-point not in } \{v_1, \ldots, v_\ell\}. \tag{13}$$

To verify if graph $\hat{H} \otimes \ell$ has a hamiltonian path and eventually find it, we can proceed as in Section 3. Indeed, considering that $\hat{H} \otimes \ell = \hat{G}'(\hat{G}_1, \ldots, \hat{G}_{\hat{n}}, I_\ell)$, and that $ham(I_\ell) = \ell$ and $\mathcal{P}_{ham(I_\ell)} = I_\ell$, then given the values $ham(\hat{G}_i), |V(\hat{G}_i)|$ and the set $\mathcal{P}_{ham(\hat{G}_i)}$, for each

$i \in \{1, \ldots, \hat{n}\}$, we can consider the corresponding ILP as in Section 3.1 choosing $B_{\hat{G}'} = \emptyset$ and $r = \hat{n}+1$ (i.e., $\hat{G}_r = I_\ell$). If the ILP admits a solution, we can construct the hamiltonian path P of $\hat{H} \otimes \ell = \hat{G}'(\hat{G}_1, \ldots, \hat{G}_{\hat{n}}, I_\ell)$ following the construction in Section 3.2 choosing any vertex in $\hat{G}_r = I_\ell$ as end-point (i.e., root). Finally, we notice that everything was proved in Section 3.2 holds also in this case (i.e., $\hat{G}'(\hat{G}_1, \ldots, \hat{G}_{\hat{n}}, I_\ell)$, $B_{\hat{G}'} = \emptyset$ and $\hat{G}_r = I_\ell$) and that, as in Section 3.4, it can be proved that if there exists a hamiltonian path of $\hat{G}'(\hat{G}_1, \ldots, \hat{G}_{\hat{n}}, I_\ell)$ with an end-point in I_ℓ then exists a solution (x, y) of the corresponding ILP (the same arguments used in the proof of Lemma 10 holds rooting the hamiltonian path at the end-point in I_ℓ).

The minimum ℓ for which (13) occurs, gives $ham(\hat{H})$ and also the hamiltonian path P of $\hat{H} \otimes ham(\hat{H})$ with one end-point in $I_{ham(\hat{H})}$. The arguments used in Section 2.3 allows to have the time complexity of this computation

$$ham(\hat{H})\ [O(\hat{n}\sqrt{\hat{n}+\hat{m}})^{(1+o(1))(3\hat{n}+2\hat{m}+1)} + O(\hat{n}(\hat{n}+\hat{m}))] + O(|V(\hat{H})|^2).$$

Obviously, the subgraph of P induced by $V(\hat{H})$ will return the partition in $ham(\hat{H})$ disjoint paths of $\hat{H}$, $\mathcal{P}_{ham(\hat{H})}$.

We stress that $ham(\hat{H}) = 1$ iff the graph $\hat{H}$ has a hamiltonian path.

References

1 J. Baste and D. Watel. An fpt algorithm for node-disjoint subtrees problems parameterized by treewidth. *The Computer Journal*, 2022.

2 J.-C. Bermond, L. Gargano, S. Perénnes, A.A. Rescigno, and U. Vaccaro. Optimal time data gathering in wireless networks with multidirectional antennas. *Theoretical Computer Science*, 509:122–139, 2013.

3 J.-C. Bermond, L. Gargano, and A.A. Rescigno. Gathering with minimum delay in tree sensor networks. In *Proceedings of 15th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2008) Alexander A. Shvartsman and Pascal Felber Eds., Lecture Notes in Computer Science*, volume 5058, pages 262–276, 2008.

4 J.-C. Bermond, L. Gargano, and A.A. Rescigno. Gathering with minimum completion time in sensor tree networks. *Journal of Interconnection Networks*, 11:1–33, 2010.

5 F. Carrabs, R. Cerulli, M. Gaudioso, and M. Gentili. Lower and upper bounds for the spanning tree with minimum branch vertices. *Computational Optimization and Applications*, 56(2):405–438, 2013.

6 C. Cerrone, R. Cerulli, and A. Raiconi. Relations, models and a memetic approach for three degree-dependent spanning tree problems. *European Journal of Operational Research*, 232(3):442–453, 2014.

7 R. Cerulli, M. Gentili, and A. Iossa. Bounded-degree spanning tree problems: models and new algorithms. *Computational Optimization and Applications*, 42(3):353–370, 2009.

8 V. M. Chimani and J. Spoerhase. Approximating spanning trees with few branches. *Theory of Computing Systems*, 56(1):181–196, 2015.

9 F.V. Fomin, P.A. Golovach, D. Lokshtanov, and S. Saurabh. Clique-width: on the price of general. In *Proceedings of the 2009 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), Claire Mathieu Ed.*, pages 825–834, 2009.

10 J. Gajarský, M. Lampis, and S. Ordyniak. Parameterized algorithms for modular-width. In *Proceedings of 8th International Symposium on Parameterized and Exact Computation (IPEC 2013), Gregory Gutin and Stefan Szeider Eds., Lecture Notes in Computer Science*, volume 8246, pages 163–176, 2013.

11 L. Gargano, M. Hammar, P. Hell, L. Stacho, and U. Vaccaro. Spanning spiders and light splitting switches. *Discrete Mathematics*, 285(1):83–95, 2004.

12 L. Gargano, P. Hell, L.Stacho, and U. Vaccaro. Spanning trees with bounded number of branch vertices. In *Proceedings of 29th International Colloquium on Automata, Languages and Programming (ICALP 2002), Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan J. Eidenbenz,Ricardo Conejo, Eds., Lecture Notes in Computer Science*, volume 2380, pages 355–365, 2002.

13 L. Gargano and A. A. Rescigno. Collision-free path coloring with application to minimum-delay gathering in sensor networks. *Discrete Applied Mathematics*, 157(8):1858–1872, 2009.

14 L. Gargano and A. A. Rescigno. Complexity of conflict-free colorings of graphs. *Theoretical Computer Science*, 566:39–49, 2015.

15 L. Gargano and A. A. Rescigno. Spanning trees with few branch vertices in graphs of bounded neighborhood diversity. In *Proceedings of 2023 International Colloquium on Structural Information and Communication Complexity (SIROCCO 2023), Sergio Rajsbaum, Alkida Balliu, Joshua J. Daymude, Dennis Olivetti Eds., Lecture Notes in Computer Science*, volume 13892, pages 502–519, 2023.

16 D. Gozupek, S. Buhari, and F. Alagoz. A spectrum switching delay-aware scheduling algorithm for centralized cognitive radio networks. *IEEE Transactions on Mobile Computing*, 12:1270–1280, 2013.

17 K. Jansen and L. Rohwedderb. On integer programming, discrepancy, and convolution. *Mathematics of Operations Research*, pages 1–15, 2023.

18 M. Landete, A. Marín, and J. L. Sainz-Pardo. Decomposition methods based on articulation vertices for degree-dependent spanning tree problems. *Computational Optimization and Applications*, 68:749–773, 2017.

19 A. Marin. Exact and heuristic solutions for the minimum number of branch vertices spanning tree problem. *European Journal of Operational Research*, 245(3):680–689, 2015.

20 H. Matsuda, K. Ozeki, and T. Yamashita. Spanning trees with a bounded number of branch vertices in a claw-free graph. *Graphs and Combinatorics*, 30:429–437, 2014.

21 R. A. Melo, P. Samer, and S. Urrutia. An effective decomposition approach and heuristics to generate spanning trees with a small number of branch vertices. *Computational Optimization and Applications*, 65(3):821–844, 2016.

22 A. Rossi, A. Singh, and S. Shyam. Cutting-plane-based algorithms for two branch vertices related spanning tree problems. *Optimization and Engineering*, 15:855–887, 2014.

23 G. Salamon. Spanning tree optimization problems with degree-based objective functions. In *Proceedings of 4th Japanese–Hungarian Sym. on Discrete Math. and Its Applications*, pages 309–315, 2005.

24 N. Shami and M. Rasti. A joint multi-channel assignment and power control scheme for energy efficiency in cognitive radio networks. In *Proceedings of 2016 IEEE Wireless Communications and Networking Conference (WCNC 2016)*, pages 1–6, 2016.

25 R. M. A. Silva, D. M. Silva, M. G. C. Resende, G. R. Mateus, J. F. Gon¸calves, and P. Festa. An edge-swap heuristic for generating spanning trees with minimum number of branch vertices. *Optimization Letters*, 8(4):1225–1243, 2014.

26 S. Silvestri, G. Laporte, and R. Cerulli. A branch-and-cut algorithm for the minimum branch vertices spanning tree problem. *Comput. Oper. Res.*, 81:322–332, 2017.

27 S. Sundar, A. Singh, and A. Rossi. New heuristics for two bounded-degree spanning tree problems. *Information Sciences*, 195:226–240, 2012.

28 M. Tedder, D.G. Corneil, M. Habib, and C. Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *Proceedings of 35th International Colloquium on Automata, Languages and Programming (ICALP 2008), Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, Igor Walukiewicz Eds., Lecture Notes in Computer Science*, volume 5125, pages 634–645, 2008.