Parameterized Leaf Power Recognition via Embedding into Graph Products

David Eppstein[1]
Computer Science Department, University of California, Irvine, USA
eppstein@uci.edu

Elham Havvaei
Computer Science Department, University of California, Irvine, USA
ehavvaei@uci.edu

──── **Abstract** ────

The k-leaf power graph G of a tree T is a graph whose vertices are the leaves of T and whose edges connect pairs of leaves at unweighted distance at most k in T. Recognition of the k-leaf power graphs for $k \geq 6$ is still an open problem. In this paper, we provide an algorithm for this problem for sparse leaf power graphs. Our result shows that the problem of recognizing these graphs is fixed-parameter tractable when parameterized both by k and by the degeneracy of the given graph. To prove this, we describe how to embed the leaf root of a leaf power graph into a product of the graph with a cycle graph. We bound the treewidth of the resulting product in terms of k and the degeneracy of G. As a result, we can use methods based on monadic second-order logic (MSO_2) to recognize the existence of a leaf power as a subgraph of the product graph.

1 Introduction

Leaf powers are a class of graphs that were introduced in 2002 by Nishimura, Ragde and Thilikos [29], extending the notion of graph powers. For a graph G, the kth power graph G^k has the same set of vertices as G but a different notion of adjacency: two vertices are adjacent in G^k if there is a path of at most k edges between them in G. The leaf powers are defined in the same way from trees, but only including the leaves of the trees as vertices. The kth leaf power of a tree T has the leaves of T as its vertices, with two vertices adjacent in the leaf power if there is a path of at most k edges between them in T. A given graph G is a k-leaf-power graph when there exists a tree T for which G is the kth leaf power. In this case, T is called the *k-leaf root* of G. For example, Figure 1 shows a 3-leaf power alongside its 3-leaf root. Nishimura et al., further, derived the first polynomial-time algorithms to recognize k-leaf powers for $k = 3$ and $k = 4$ [29].

The problem of recognizing leaf powers arises as a formalization of a problem in computational biology, the reconstruction of evolutionary history and evolutionary trees from information about the similarity between species [12, 21]. In this problem, the common ancestry of different species can be represented by an evolutionary or phylogenetic tree, in
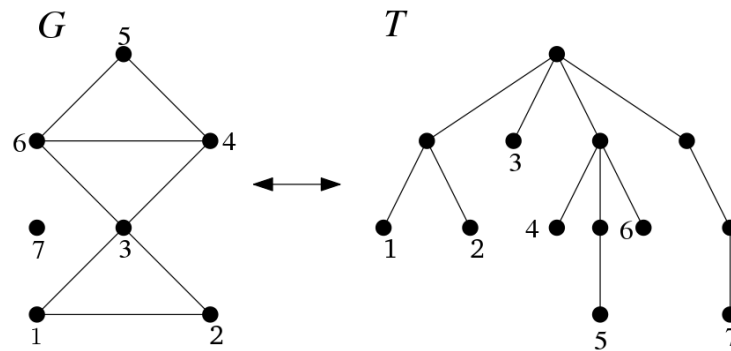
Figure 1 A 3-leaf power graph G and its 3-leaf root T.

which each vertex represents a species and each edge represents a direct ancestry relation between two species. We only have full access to living species, the species at the leaves of the tree; the other species in the tree are typically long-extinct, and may be represented physically only through fossils or not at all. If we suppose that we can infer, from observations of living species, which ones are close together (within some number k of steps in this tree) and which others are not, then we could use an algorithm for leaf power recognition to infer a phylogenetic tree consistent with this data.

1.1 New Results

In this paper, we prove that the k-leaf powers of degeneracy d can be recognized in time that is fixed-parameter tractable when parameterized by k and d. Here, the degeneracy of a graph is the maximum, over its subgraphs, of the minimum degree of any subgraph. Our result provides an algorithm whose running time is polynomial (in fact linear) in the size of its input graph, multiplied by a factor that depends non-polynomially on k and d.

It is known that the k-leaf powers have bounded clique-width for bounded k [24]. A wide class of graph problems (those expressible in a version of monadic second order logic quantifying over only vertex sets, MSO_1) can be solved in fixed-parameter time for graphs of bounded clique-width, via Courcelle's theorem [16]. However we have been unable to express the recognition of leaf powers in MSO_1. Instead, our algorithm uses a more powerful version of monadic second order logic allowing quantification over edge sets, MSO_2. The graphs of bounded clique-width and bounded degeneracy have bounded treewidth, allowing us to apply a form of Courcelle's theorem for MSO_2 and for graphs of bounded treewidth.

However, there is an additional complication that makes it tricky to apply these methods to leaf power recognition. The tree that we wish to find, for which our given input graph is a leaf power, will in general include vertices and edges that are not part of the input, but MSO_2 can only quantify over subsets of the existing vertices and edges of a graph, not over sets of vertices and edges that are not subsets of the input. To work around this problem, we apply Courcelle's theorem not to the given graph G itself, but to a *product graph* $G \boxtimes C_k$ where C_k is a k-vertex cycle graph. We prove that the leaf root (the tree for which G is a leaf power, if it is one) can be embedded as a subgraph of this product, that it can be recognized by an MSO_2 formula applied to this product, and that this product has bounded treewidth whenever G is a k-leaf power of bounded degeneracy. In this way we can recognize G as a leaf power, not by applying Courcelle's theorem to G, but by applying it to the product graph.
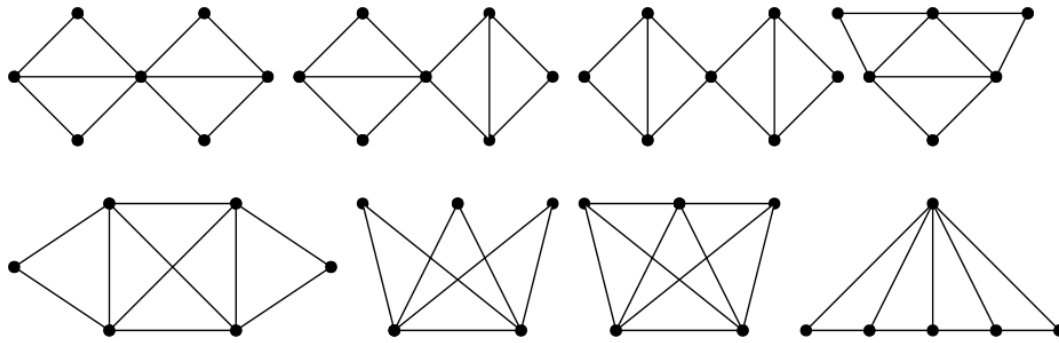
Figure 2 A graph is a 4-leaf power if and only if it is chordal and does not contain any of the graphs above as a subgraph.

Thus, our algorithm combines the following ingredients:

- Our embedding of the k-leaf root as a subgraph of the product graph $G \boxtimes C_k$.
- Our logical representation of k-leaf roots as subgraphs of product graphs.
- Courcelle's theorem, which provides general-purpose algorithms for testing MSO_2 formulas on graphs of bounded treewidth.
- The fact that the k-leaf powers, for bounded values of k, have bounded clique-width.
- The fact that graphs of bounded clique-width and bounded degeneracy also have bounded treewidth.
- The fact that, by taking a product with a graph of bounded size, we preserve the bounded treewidth of the product.

Our algorithm runs in fixed-parameter tractable time when parameterized by k and the degeneracy d of the given input graph. In particular, it runs in linear-time when k and d are both constant.

Our result provides the first known efficient algorithm for recognizing k-leaf powers for $k \geq 6$, for graphs of bounded degeneracy. More generally, our method of embedding into graph products appears likely to apply to other graph problems involving network design (the addition of edges to an existing graph, rather than the identification of a special subgraph of the input). In the case we apply this method to leaf power recognition, we expect that it should be possible to translate our MSO_2 formula over the product graph into a significantly more complicated MSO_2 formula over the input graph, but the method of embedding into graph products considerably simplifies our task of designing a logical formula for our problem.

1.2 Related Work

Polynomial-time algorithms are known for recognizing k-leaf powers for $k \geq 5$.

- A graph is a 2-leaf power if it is a disjoint union of cliques, so this class of graphs is trivial to recognize.
- There exist various ways to characterize 3-leaf powers [29, 7, 18, 30], some of which lead to efficient algorithms. For instance, one way to determine if a graph is a 3-leaf power is to check whether it is bull-, dart- and gem-free and chordal [18]. The chordal graphs have a known recognition algorithm, and testing for the existence of any of the other forbidden induced subgraphs is polynomial, because they all have bounded size.
- Similarly, there are various known ways to characterize 4-leaf powers [29, 30, 19, 9]. One is that graph is a 4-leaf power if and only if it is chordal and does not contain any
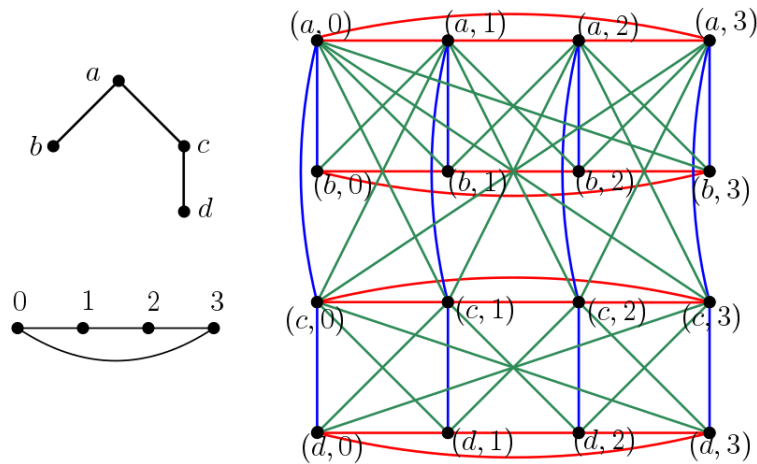
Figure 3 The graph on the right is the strong product of a four-vertex path graph (top left) and a four-vertex cycle graph (bottom left). The colors indicate the partition of the edges into vertical, horizontal, and diagonal subsets.

of the graphs depicted in Figure 2 as induced subgraphs [30]. Again, this leads to a polynomial-time recognition algorithm, because all of these graphs have bounded size.

- k-leaf powers can be recognized in polynomial time if the $(k-2)$- Steiner root problem can be solved in polynomial time. Chang and Ko in 2007, provided a linear-time algorithm recognition for 3-Steiner root problem [11]. This implies that 5-leaf powers can be recognized in linear time.

Polynomial-time structural characterization of k-leaf powers for $k \geq 6$ is still an open problem. Nonetheless, Brandstädt et al. provided a characterization for the distance-hereditary 5-leaf powers [8].

Throughout the literature, there exist many structural characterizations of leaf powers which provide potentially useful insight into this class of graphs. It is known, for instance, that all leaf powers are strongly chordal, but the converse is not always true. Further, Kennedy et al. showed that strictly chordal graphs are always k-leaf powers for $k \geq 4$; these are the chordal graphs that are also, dart- and gem-free. They provided a linear-time algorithm to construct k-leaf roots of strictly chordal graphs [27].

For all $k \geq 2$, every k-leaf power is also a $(k+2)$-leaf power. A $(k+2)$-leaf root of any k-leaf-power can be obtained from its k-leaf root, by subdividing all edges incident to leaves. However, the problems of recognizing k-leaf powers for different values of k do not collapse: for all $k \geq 4$, there exists a k-leaf power which is not a $(k+1)$-leaf power [10].

2   Preliminaries

2.1   Definitions

Throughout this paper, we let $G(V, E)$ denote a simple undirected graph (typically, the input to the leaf power recognition problem). If u and v are two vertices in V that are adjacent in G, we let $e(u, v)$ denote the edge connecting them.

The strong product of graphs G_1 and G_2, denoted as $G_1 \boxtimes G_2$, is a graph whose vertices are ordered pairs of a vertex from G_1 and a vertex from G_2. In it, two distinct vertices (u_1, u_2) and (v_1, v_2) are adjacent if and only if for all $i \in \{1, 2\}$, $u_i = v_i$ or u_i and v_i are adjacent in G_i. Figure 3 shows an example, the strong product of a four-vertex path graph

with a four-vertex cycle graph. When we construct a strong product, we will classify the edges of the product into three subsets:

- We call an edge from (u_1, u_2) to (v_1, v_2) a *vertical edge* if $u_2 = v_2$. The edges of this type form $|V(G_2)|$ disjoint copies of G_1 as subgraphs of the product.
- We call an edge from (u_1, u_2) to (v_1, v_2) a *horizontal edge* if $u_1 = v_1$. The edges of this type form $|V(G_1)|$ disjoint copies of G_2 as subgraphs of the product.
- We call the remaining edges, for which $u_1 \neq v_1$ and $u_2 \neq v_2$, *diagonal edges*. The subgraph composed of the diagonal edges forms a different kind of graph product, the *tensor product* $G_1 \times G_2$.

We may think of these three edge sets as forming an (improper) edge coloring of the product graph. In Figure 3 these edge sets are colored blue, red and green, respectively.

2.2   Graph Parameters

One of the simplest ways of parameterizing sparse graphs is by their *degeneracy*. The degeneracy $d(G)$ of a graph G is the smallest number such that every nonempty subgraph of G contains at least one vertex of degree at most $d(G)$. It may be computed in linear-time by a greedy algorithm that repeatedly removes the minimum-degree vertex and records the largest degree seen among the vertices at the time they are removed [28].

The notion of *treewidth*, a more complicated graph sparsity parameter, was first introduced by Bertelé and Brioschi [3] and Halin [25] and later rediscovered by Robertson and Seymour [31]. One way to define treewidth is to use the concept of tree decomposition. A tree decomposition of graph G consists of a tree T where each vertex $X_i \in T$ (called a bag) is a subset of vertices of G . This tree and its bags are required to satisfy the following properties:

- For each edge $e(u, v)$ in G, there exists a bag in T containing both u and v; and
- For each vertex v in G, the bags containing v form a nonempty connected subtree of T.

The width of a tree decomposition is the size of its largest bag, minus one. The treewidth of a graph is defined as the minimum width achieved over all tree decompositions of the graph. Bounded treewidth graphs are especially interesting from an algorithmic point of view. Many well-known NP-complete problems have linear-time algorithms on graphs of bounded treewidth [4].

Another related graph parameter, *clique-width*, was introduced by Courcelle et al. to characterize the structural complexity of graphs [16]. The clique-width of a graph G is the minimum number of labels necessary to construct G by means of four graph operations: creation of a new vertex with a label, vertex disjoint union of labeled graphs, insertion of an edge between two vertices with specified labels and relabeling of vertices. Relevantly for us, Courcelle et al. showed that unit interval graphs are of unbounded clique-width. A graph is an interval graph if and only if all its vertices can be mapped into intervals on a straight line such that two vertices are adjacent when the corresponding intervals intersect each other. In the unit interval graphs, each interval has a unit length. As shown by Brandstädt et al., unit interval graphs belong to the class of leaf powers, which implies that leaf powers also have unbounded clique-width [5, 6]. However, Gurski and Wanke proved that for a fixed k, k-leaf powers have a clique-width of at most $k + \max(\lfloor \frac{k}{2} \rfloor - 2, 0)$ [24]. Therefore, when seeking the complexity of recognizing leaf powers, parameterized by k, we may restrict our attention to graphs of bounded clique-width.

These three properties are defined differently to each other, and may have significantly different values. For instance, the complete bipartite graph $K_{n,n}$ has clique-width two but treewidth and degeneracy n, and the $n \times n$ grid graph has degeneracy two but clique-width and treewidth $\Omega(n)$. Nevertheless, there is an important relation between them: in the

graphs of bounded degeneracy and bounded clique-width, the treewidth is also bounded. More specifically, if a graph has clique-width k and contains no complete bipartite subgraph $K_{t,t}$, then it has treewidth at most $3k(t-1)-1$. Since such a subgraph would force the degeneracy to be at least t, it follows that the graphs of degeneracy d and clique-width k have treewidth at most $3kd-1$ [23].

2.3 Courcelle's Theorem

By considering graphs as logical structures, their properties can be expressed in first-order and second-order logic. In first-order logic, graph properties are expressed as logical formulas wherein the variables range over vertices and the predicates include equality and adjacency relations. Second-order logic is an extension of first-order logic with the power to quantify over relations. Particularly, many natural graph properties can be described in monadic second-order logic, which is a restriction of second-order logic in which only unary relations (sets of vertices or edges) are allowed [15].

There exist two variations of monadic second-order logic: MSO_1 and MSO_2. In MSO_1, quantification is allowed only over sets of vertices, while MSO_2 allows quantification over both sets of vertices and sets of edges. MSO_2 is strictly more expressive; there are some properties, such as Hamiltonicity [14], which are expressible in MSO_2 but not in MSO_1. A graph property is MSO_2-*expressible* if there exists an MSO_2 formula to express it, in which case the corresponding class of graphs becomes MSO_2-*definable*.

The connection between treewidth and monadic second-order logic is given by Courcelle's theorem, according to which every property definable in monadic second-order logic can be tested in linear time on graphs of bounded treewidth [13]. Later, Courcelle et al. extended this theorem to the class of graphs with bounded clique-width when the underlying property is MSO_1-definable [17]. In our application of Courcelle's theorem, we will use an MSO_2 formula with a free variable HORIZONTAL, an edge set, which we will use to pass to the formula certain information about the structural decomposition of the graph it is operating on. This extension of Courcelle's theorem to formulas with a constant number of additional free variables, whose values are assigned through some extra-logical process prior to applying the theorem, is non-problematic and standard.

However, even in MSO_2, it is only possible to quantify over subsets of vertices and edges that belong to the graph to which the logical formula is applied. Much of the difficulty of the leaf power problem rests in this restriction. If we could quantify over edges and vertices that were not already present, we could construct a formula that asserts the existence of sets of vertices and edges forming a leaf root of a given graph, and then add clauses to the formula that ensure that the quantified sets describe a valid leaf root. However, we are not allowed such quantification, because in general the leaf root has vertices and edges that do not belong to our input graph. To apply Courcelle's theorem to leaf power recognition, we must instead find a way to express the property of being a leaf power using only quantification over subsets of vertices and edges of the graph to which we apply the theorem. For this reason, the problem of leaf power recognition forms an important test case for the ability to express graph problems in MSO logic.

A problem is *fixed-parameter tractable* with respect to a parameter x of the input if the problem can be solved in time $f(x)n^{O(1)}$ where n is the size of the input, f is a computable function of x (independent of n), and the exponent of n in the $O(1)$ term is independent of x. Courcelle's theorem is the foundation of many fixed-parameter tractable algorithms [2, 22, 20, 26], as it proves that properties expressible in MSO_1 or MSO_2 are fixed-parameter tractable with respect to the clique-width or treewidth (respectively) of the input graph.
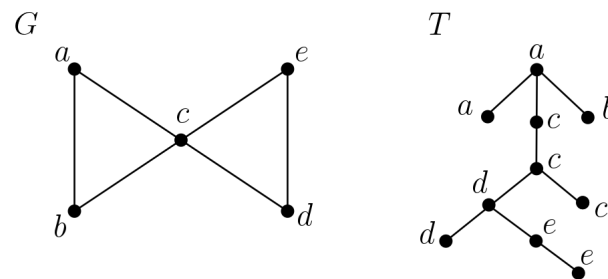
## 3 Embedding Leaf Roots Into Graph Products

Let G be a k-leaf power graph, and T be a k-leaf root of G. If G is not connected, we can handle each of its connected components independently; in this way, we can assume from now on, without loss of generality that G is a connected graph with at least three vertices, and that T is a leaf root chosen arbitrarily among the possible k-leaf roots of T. It follows from these assumptions that T is a tree, because every edge in G must be represented by a path in T. Because T has at least three leaves, it has at least one interior node; we choose one of these nodes arbitrarily to be the root of T. Additionally, every vertex or edge of T participates in a path of length at most k between two leaves, representing an edge of G. For, if some vertices and edge do not participate in these paths, removing all non-participating vertices and edges from T would produce a smaller leaf root, without creating any new leaves. But this removal would disconnect pairs of leaves on the opposite sides of any removed edge, contradicting the assumption that G is connected.

As the first step of the embedding, we label the vertices of T with the names of vertices in G. Each vertex of T will get a label in this way; some labels will be used more than once. In particular, we label each leaf of T by the vertex of G represented by that leaf. Then, as shown in Figure 4, we give each non-leaf node of T the same label as its closest leaf. If there are two or more closest leaves, we choose which one to use as the label arbitrarily among the labels already applied to the child of the given interior node. In this way, when the same label appears more than once, the tree nodes having that label form a connected path in T.

As we now show, these labels, together with the depths of the nodes modulo k, can be used to embed the k-leaf root T into the strong product $G \boxtimes C_k$, where C_k denotes a k-vertex cycle graph.

▶ **Lemma 1.** *If G is a connected k-leaf power graph on three or more vertices, and T is any k-leaf root of G, then T can be embedded as a subgraph of the strong product $G \boxtimes C_k$. Additionally, the embedding can be chosen in such a way that each horizontal cycle in the strong product (the product of a vertex v of G with C_k) contains exactly one leaf of the embedded copy of T, the leaf representing v.*

Proof. We map a vertex u of T to the pair (v, i) where v is the label assigned to u (the name of a vertex in G) and i is the depth of u (its distance from the root of T), taken modulo k. This pair is one of the vertices of the strong product, so we have mapped vertices of T into vertices of the strong product. An example of such embedding can be seen in Figure
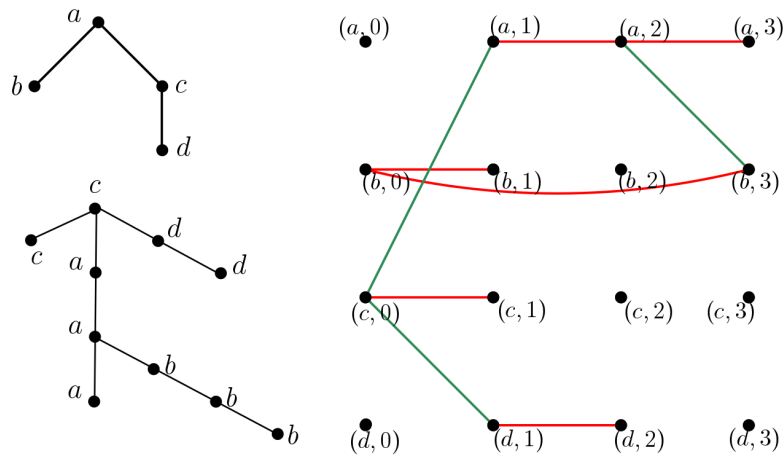
Figure 5 The graph on the bottom left is a 4-leaf root T of graph G (top left). T can be embedded in the strong product $G \boxtimes C_4$ (right), by mapping each vertex u of T to the pair (v, i) where v is the label of u and i is the depth of u (modulo k).

Figure 5. Because G is assumed to be connected, each node of T participates in at least one path of length at most k between two leaves of T, representing an adjacency of G; it follows that the label for each node of T is at most $k - 1$ steps away from the node, and that each path of same-labeled nodes in T has length at most $k - 1$. As a consequence, when we take depths modulo k, none of these paths can wrap around the cycle and cover the same vertex of the product graph more than once. That is, our mapping from T to $G \boxtimes C_k$ is one-to-one. Because each leaf of T is labeled with the vertex of G that it represents, this mapping has the property described in the lemma, that each horizontal cycle in the strong product contains exactly one leaf of the embedded copy of T, the leaf representing the vertex whose product with C_k forms that particular horizontal cycle.

We must also show that this mapping from T to $G \boxtimes C_k$ maps each pair of vertices that are adjacent in T into a pair of vertices that are adjacent in $G \boxtimes C_k$. Recall that adjacency in $G \boxtimes C_k$ is the conjunction of two conditions: two vertices in the product are adjacent if their first coordinates are equal or adjacent in G and their second coordinates are equal or adjacent in C_k. Because every two adjacent vertices in T have depths that differ by one, the second coordinates of their images in the product will always be adjacent in C_k. It remains to show that, when two vertices are adjacent in T, their images in the product have first coordinates that are equal or adjacent in G. That is, the labels of the two adjacent vertices in T should be equal or adjacent.

Rephrasing what we still need to show, it is the following: whenever two adjacent vertices in T have different labels, those labels represent adjacent vertices in G.

To see that this is true, consider two adjacent vertices u_1 and its parent u_2 in T, labeled by two different vertices v_1 and v_2 in G. As we already stated at the start of this section, the assumption of the lemma that G is connected implies that edge $u_1 u_2$ in T participates in at least one path P of length at most k between two leaves, corresponding to an adjacency in G. But because v_1 and v_2 are represented by the closest leaves to u_1 and u_2 (respectively) the length of the path in T between the leaves representing v_1 and v_2 must be at most equal to the length of P. Therefore, there is a path of length at most k between the leaves representing v_1 and v_2, so v_1 and v_2 are adjacent in the k-leaf power G, as required.   ◄

Based on this embedding, we can prove the following characterization of leaf powers, which we will use in our application of Courcelle's theorem to the problem. It is important, for this characterization, that we express everything intrinsically in terms of the properties of the product graph $G \boxtimes C_k$, its edge coloring, and its subgraphs, without reference to the given graph G.

▶ **Lemma 2.** *A given connected graph G on three or more vertices is a k-leaf power if and only if the product $G \boxtimes C_k$ has a subgraph T with the following properties:*

1. *T is 1-degenerate (i.e., a forest).*
2. *Every vertex of $G \boxtimes C_k$ is connected by horizontal edges of the product to exactly one leaf of T.*
3. *Two vertices of $G \boxtimes C_k$ are the endpoints of a non-horizontal edge of the product if and only if the corresponding leaves of T (given according to Property 2) are the distinct endpoints of a path of length at most k in T.*

Proof. A subgraph obeying these properties is a forest (Property 1), whose leaves can be placed into one-to-one correspondence with the vertices of G (Property 2, using the fact that the horizontal cycles of the product correspond one-for-one with vertices of G). It has a path of length at most k between two leaves if and only if the corresponding vertices of G are adjacent (Property 3). So if it exists, it is a k-leaf root of G and G is a k-leaf power.

In the other direction, if G is a connected k-leaf power, let T be a k-leaf root of G, embedded according to Lemma 1. Then T is a subtree of the product graph (Property 1), with exactly one leaf for each horizontal cycle (Property 2), that forms a k-leaf root of G (Property 3). So when G is a k-leaf power, a subgraph T obeying the properties of the lemma exists. ◀

4 Logical Expression

In this section, we describe how to express the components of Lemma 2, our characterization of the products $G \boxtimes C_k$ that contain a k-leaf root of G, in monadic second-order logic. Our logical formula will involve a free variable HORIZONTAL, the subset of edges of the given graph (assumed to be of the form $G \boxtimes C_k$) that are horizontal in the product (that is, edges that connect two copies of the same vertex in G). We will also assume that V and E refer to the vertices and edges of the graph $G \boxtimes C_k$. In our logical formulas, we will express the type of each quantified variable (whether it is a vertex, edge, set of vertices, or set of edges) by annotating its quantifier with a membership or subset relation. For instance, "$\forall x \in V : \ldots$" quantifies x as a vertex variable. We will express the incidence predicate between an edge e and a vertex v (true if v is an endpoint of e, false otherwise) by $e \multimap v$. Because our formulas will also use equality as a predicate, we will express the equality between names of formulas and their explicit logical formulation using a different symbol, $\equiv$. In our formulas, predicates (equality, incidence, and adjacence) will be considered to bind more tightly than logical connectives, allowing us to omit parentheses in many cases.

A subgraph of the given graph may be represented by its set S of edges. In this representation, adjacency between two vertices a and b may be expressed by the formula

$$\text{ADJACENT}(a, b, S) \equiv \exists e \in S : (e \multimap a \wedge e \multimap b).$$

The following formula expresses the property that the neighbors of vertex ℓ in subgraph S include at most one vertex from a set X:

$$\text{LEAF}(\ell, X, S) \equiv \forall c, d \in X : \Big(\big(\text{ADJACENT}(\ell, c, S) \wedge \text{ADJACENT}(\ell, d, S) \big) \to c = d \Big).$$

This allows us to express the acyclicity of a subgraph S in terms of 1-degeneracy: every nonempty subset Y of vertices contains a leaf.

$$\text{ACYCLIC}(S) \equiv \forall X \subset V : (\exists x \in X) \rightarrow \exists \ell \in X : \text{LEAF}(\ell, X, S).$$

This already allows us to express the first condition of Lemma 2.

We will also use a predicate for whether two vertices p and q are connected by horizontal edges. This is true if there is no cut $(C, V \setminus C)$ that separates them.

$$\text{ALIGNED}(p, q) \equiv \forall C \subset V : \big(p \in C \wedge \neg(q \in C)\big) \rightarrow$$
$$\exists h \in \text{HORIZONTAL} : \exists y, z \in V : \big(y \in C \wedge \neg(z \in C) \wedge h \multimap y \wedge h \multimap z\big).$$

This allows us to express a predicate for the property that vertex ℓ is a leaf of subgraph S on the same horizontal level as another vertex v (that is, ℓ is the representative leaf for v's level):

$$\text{REPRESENTATIVE}(v, \ell, S) \equiv \text{LEAF}(\ell, V, S) \wedge \text{ALIGNED}(v, \ell).$$

The second part of Lemma 2 is that every level has exactly one representative leaf:

$$\text{REPRESENTED}(S) \equiv \big(\forall v \in V : \exists \ell \in V : \text{REPRESENTATIVE}(v, \ell, S)\big) \wedge$$
$$\big(\forall v, \ell_1, \ell_2 \in V : \big(\text{REPRESENTATIVE}(v, \ell_1, S) \wedge \text{REPRESENTATIVE}(v, \ell_2, S)\big) \rightarrow \ell_1 = \ell_2\big)$$

Unlike for the previous formulas, there is no way of expressing the existence of a path of length k from u to v in subgraph S, for a variable k, in MSO_2. We need a different formula PATH_k for each k. We do not require these paths to be simple, as this would only complicate the formula without simplifying our use of it. However it is essential for our application to the third condition of Lemma 2 that we require our paths to have distinct endpoints.

$$\text{PATH}_k(u, v, S) \equiv \exists w_1, w_2, \ldots w_{k-1} \in V : \exists e_1, e_2, \ldots e_k \in S :$$
$$\neg(u = v) \wedge e_1 \multimap u \wedge e_1 \multimap w_1 \wedge e_2 \multimap w_1 \wedge \cdots \wedge e_k \multimap w_{k-1} \wedge e_k \multimap v.$$

Other than the inequality of the two endpoints, this formula allows repetitions of vertices and edges within each path. In particular, it allows w_i and w_{i+1} to be equal to each other, repeating one endpoint of an edge twice and omitting the other endpoint. Because we allow repetitions in this way, this formulation of the PATH predicate has the following convenient property:

▶ **Lemma 3.** *For all $k \geq 1$ and all u, v, and S, we have that*

$$\text{PATH}_k(u, v, S) \rightarrow \text{PATH}_{k+1}(u, v, S).$$

Proof. Let $w_1, \ldots w_{k-1}$ and $e_1, \ldots e_k$ be the vertices and edges witnessing the truth of $\text{PATH}_k(u, v, S)$, let $w_k = v$, and let $e_{k+1} = e_k$. Then $w_1, \ldots, w_k$ and $e_1, \ldots, e_{k+1}$ witness the truth of $\text{PATH}_{k+1}(u, v, S)$. ◀

▶ **Corollary 4.** *Two vertices u and v of a subgraph S of a given graph obey the predicate $\text{PATH}_k(u, v, S)$ if and only if they are distinct and their distance in S is at most k.*

This allows us to express the final part of Lemma 2, the requirement that each two vertices are connected by a non-horizontal edge if and only if their representatives are connected by a short path:

$$\text{ROOT}_k(S) \equiv \forall u, v \in V : \Big(\big(\exists e \in E : e \multimap u \wedge e \multimap v \wedge \neg(e \in \text{HORIZONTAL})\big) \longleftrightarrow$$
$$\exists x, y \in V : \big(\text{REPRESENTATIVE}(u, x, S) \wedge \text{REPRESENTATIVE}(v, y, S) \wedge \text{PATH}_k(x, y, S)\big)\Big).$$

▶ **Lemma 5.** *There exists an* MSO_2 *formula that is modeled by a graph* $G \boxtimes C_k$ *and its set* HORIZONTAL *of horizontal edges exactly when* $G \boxtimes C_k$ *meets the conditions of Lemma 2.*

Proof. The formula is

$$\exists S : \big(\textsc{acyclic}(S) \wedge \textsc{represented}(S) \wedge \textsc{root}_k(S)\big).$$

A subgraph defined by a set S of its edges meets the first condition of the lemma if $\textsc{acyclic}(S)$ is true, it meets the second condition of the lemma if $\textsc{represented}(S)$ is true, and it meets the third condition of the lemma if $\textsc{root}_k(S)$ is true.   ◀

▶ **Corollary 6.** *The property of a graph* G *being* k*-leaf power can be expressed as an* MSO_2 *formula of* $G \boxtimes C_k$ *and of the set* HORIZONTAL *of horizontal edges of this product graph.*

5 Fixed-Parameter Tractability of Leaf Powers

In order to apply Courcelle's theorem to the product graph $G \boxtimes C_k$ we need to bound its treewidth.

▶ **Lemma 7.** *If* G *has treewidth* t *and* H *has a bounded number of vertices* s *then* $G \boxtimes H$ *has treewidth at most* $s(t+1) - 1$.

Proof. Given any tree-decomposition of G with width t, we can form a decomposition of $G \boxtimes H$ by using the same tree, and placing each vertex (v, w) of $G \boxtimes H$ (where v and w are vertices of G and H respectively) into the same bag as vertex v of G. The size of the largest bag of the tree-decomposition of G is $t + 1$, so the size of the largest bag of the resulting tree-decomposition of the product graph is $s(t+1)$. The treewidth is one less than the size of the largest bag.   ◀

▶ **Corollary 8.** *If* G *has a bounded treewidth and* k *is bounded, then* $G \boxtimes C_k$ *also has bounded treewidth.*

This gives us our main theorem:

▶ **Theorem 9.** *For fixed constants* k *and* d, *it is possible to recognize in linear time (with fixed-parameter tractable dependence on* k *and* d) *whether a graph of degeneracy at most* d *is a* k*-leaf power.*

Proof. If we consider the class of graphs with bounded degeneracy d, trivially these graphs do not contain the complete bipartite graph $K_{d+1,d+1}$ as a subgraph. Gurski and Wanke proved that k-leaf powers with fixed k have bounded clique-width [24]. In separate work, they also proved that every class of graphs with bounded clique-width and no $K_{a,a}$ subgraph has bounded treewidth [23]. These results immediately imply that k-leaf powers with bounded degeneracy have bounded treewidth, and it follows from Corollary 8 that $G \boxtimes C_k$ also has bounded treewidth. Therefore, by applying Courcelle's theorem to the MSO_2 formula of Corollary 6 we obtain the result.   ◀

6 Conclusion

We have provided a fixed-parameter algorithm to recognize k-leaf powers for graphs of bounded degeneracy. Our method embeds a k-leaf root of a k-leaf power graph in the product graph of the input graph and a k-vertex cycle C_k, finds a logical characterization of the

leaf roots that are embedded in this way, and applies Courcelle's theorem to determine the existence of a subgraph of the product graph that meets our characterization.

It is perhaps of interest to observe that, although ACYCLIC and ALIGNED involve quantification over vertex sets, the only quantification over edge sets that we use is in the formula of Lemma 5. If we could eliminate all edge sets from our formula (including HORIZONTAL), and obtain an MSO_1 formula for k-leaf powers in place of the MSO_2 formula that we use, it might be possible to eliminate the dependence on degeneracy from our parameterized analysis.

Our methods of using low-treewidth supergraphs to represent vertices and edges that are not part of the input graph, and of using graph products to find these supergraphs, may be useful in other graph problems. For instance, this method would have greatly simplified the application of Courcelle's theorem in our recent work on planar split thickness [20]: a graph G has planar split thickness k if and only if $G \boxtimes K_k$ has a planar subgraph S such that, for each non-horizontal edge of the product, the endpoints of the edge are aligned with the endpoints of an edge in S. In reducing the logical complexity of problems such as these, our method also makes it more likely that faster model checkers for restricted fragments of MSO logic [1] can be applied to our problem. Additionally, and more speculatively, the reduced logical complexity of our method may help guide future efforts to solve the problem directly using dynamic programming rather than by applying Courcelle's theorem.

References

1   Max Bannach and Sebastian Berndt. Practical Access to Dynamic Programming on Tree Decompositions. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.ESA.2018.6`.

2   Michael J Bannister and David Eppstein. Crossing minimization for 1-page and 2-page drawings of graphs with bounded treewidth. In *International Symposium on Graph Drawing*, pages 210–221. Springer, 2014.

3   Umberto Bertelé and Francesco Brioschi. *Nonserial Dynamic Programming.* Academic Press, 1972.

4   Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybernet.*, 11(1-2):1–21, 1993.

5   Andreas Brandstädt and Christian Hundt. Ptolemaic graphs and interval graphs are leaf powers. In *Latin American Symposium on Theoretical Informatics*, pages 479–491. Springer, 2008.

6   Andreas Brandstädt, Christian Hundt, Federico Mancini, and Peter Wagner. Rooted directed path graphs are leaf powers. *Discrete Math.*, 310(4):897–910, 2010. `doi:10.1016/j.disc.2009.10.006`.

7   Andreas Brandstädt and Van Bang Le. Structure and linear time recognition of 3-leaf powers. *Inform. Process. Lett.*, 98(4):133–138, 2006. `doi:10.1016/j.ipl.2006.01.004`.

8   Andreas Brandstädt, Van Bang Le, and Dieter Rautenbach. A forbidden induced subgraph characterization of distance-hereditary 5-leaf powers. *Discrete Math.*, 309(12):3843–3852, 2009. `doi:10.1016/j.disc.2008.10.025`.

9   Andreas Brandstädt, Van Bang Le, and R. Sritharan. Structure and linear-time recognition of 4-leaf powers. *ACM Trans. Algorithms*, 5(1):A11:1–A11:22, 2009. `doi:10.1145/1435375.1435386`.

10 Andreas Brandstädt and Peter Wagner. On k-versus (k+ 1)-leaf powers. In *International Conference on Combinatorial Optimization and Applications*, pages 171–179. Springer, 2008.

11 Maw-Shang Chang and Ming-Tat Ko. The 3-Steiner root problem. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 109–120. Springer, 2007.

12 Zhi-Zhong Chen, Tao Jiang, and Guohui Lin. Computing phylogenetic roots with bounded degrees and errors. *SIAM J. Comput.*, 32(4):864–879, 2003. `doi:10.1137/S0097539701389154`.

13 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inform. and Comput.*, 85(1):12–75, 1990. `doi:10.1016/0890-5401(90)90043-H`.

14 Bruno Courcelle. On the expression of graph properties in some fragments of monadic second-order logic. In Neil Immerman and Phokion G. Kolaitis, editors, *Descriptive Complexity and Finite Models: Proceedings of a DIMACS Workshop, January 14–17, 1996, Princeton University*, volume 31 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 33–62. American Mathematical Society, Providence, RI, 1997.

15 Bruno Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In *Handbook of graph grammars and computing by graph transformation, Vol. 1*, pages 313–400. World Scientific, River Edge, NJ, 1997. `doi:10.1142/9789812384720_0005`.

16 Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *J. Comput. System Sci.*, 46(2):218–270, 1993. `doi:10.1016/0022-0000(93)90004-G`.

17 Bruno Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. `doi:10.1007/s002249910009`.

18 Michael Dom, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Error compensation in leaf root problems. In *International Symposium on Algorithms and Computation*, pages 389–401. Springer, 2004.

19 Michael Dom, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Extending the tractability border for closest leaf powers. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 397–408. Springer, 2005.

20 David Eppstein, Philipp Kindermann, Stephen Kobourov, Giuseppe Liotta, Anna Lubiw, Aude Maignan, Debajyoti Mondal, Hamideh Vosoughpour, Sue Whitesides, and Stephen Wismath. On the planar split thickness of graphs. *Algorithmica*, 80(3):977–994, 2018. `doi:10.1007/s00453-017-0328-y`.

21 Walter M. Fitch and Emanuel Margoliash. Construction of phylogenetic trees. *Science*, 155(3760):279–284, 1967. `doi:10.1126/science.155.3760.279`.

22 Martin Grohe. Computing crossing numbers in quadratic time. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, pages 231–236, New York, 2001. ACM. `doi:10.1145/380752.380805`.

23 Frank Gurski and Egon Wanke. The tree-width of clique-width bounded graphs without $K_{n,n}$. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 196–205. Springer, 2000.

24 Frank Gurski and Egon Wanke. The clique-width of tree-power and leaf-power graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 76–85. Springer, 2007.

25 Rudolf Halin. *S*-functions for graphs. *J. Geometry*, 8(1-2):171–186, 1976. `doi:10.1007/BF01917434`.

26 Petr Hliněný. Branch-width, parse trees, and monadic second-order logic for matroids. *J. Combin. Theory Ser. B*, 96(3):325–351, 2006. `doi:10.1016/j.jctb.2005.08.005`.

27   William Kennedy, Guohui Lin, and Guiying Yan. Strictly chordal graphs are leaf powers. *J. Discrete Algorithms*, 4(4):511–525, 2006. `doi:10.1016/j.jda.2005.06.005`.

28   David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, 1983. `doi:10.1145/2402.322385`.

29   Naomi Nishimura, Prabhakar Ragde, and Dimitrios M. Thilikos. On graph powers for leaf-labeled trees. *J. Algorithms*, 42(1):69–108, 2002. `doi:10.1006/jagm.2001.1195`.

30   Dieter Rautenbach. Some remarks about leaf roots. *Discrete Math.*, 306(13):1456–1461, 2006. `doi:10.1016/j.disc.2006.03.030`.

31   Neil Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. `doi:10.1016/0196-6774(86)90023-4`.