Hamiltonian paths in the square of a tree

Jakub Radoszewski[1] **and Wojciech Rytter**[1,2⋆]

[1] Department of Mathematics, Computer Science and Mechanics,
University of Warsaw, Warsaw, Poland
[jrad,rytter]@mimuw.edu.pl
[2] Faculty of Mathematics and Informatics,
Copernicus University, Toruń, Poland

Abstract. We introduce a new family of graphs for which the Hamiltonian path problem is non-trivial and yet has a linear time solution. The square of a graph $G = (V, E)$, denoted as G^2, is a graph with the set of vertices V, in which two vertices are connected by an edge if there exists a path of length at most 2 connecting them in G. Harary & Schwenk (1971) proved that the square of a tree T contains a Hamiltonian cycle if and only if T is a caterpillar, i.e., it is a single path with several leafs connected to it. Our first main result is a simple graph-theoretic characterization of trees T for which T^2 contains a Hamiltonian path: T^2 has a Hamiltonian path if and only if T is a horsetail (the name is due to the characteristic shape of these trees, see Figure 1). Our next results are two efficient algorithms: linear time testing if T^2 contains a Hamiltonian path and finding such a path (if there is any), and linear time preprocessing after which we can check for any pair (u, v) of nodes of T in constant time if there is a Hamiltonian path from u to v in T^2.

Keywords: tree, square of a graph, Hamiltonian path

1 Introduction

Classification of graphs admitting Hamiltonian properties is one of the fundamental problems in graph theory, no good characterization of hamiltonicity is known. There are multiple results dealing with algorithms for finding Hamiltonian cycles in particular families of graphs, see e.g. [2].

The k-th power of a graph $G = (V, E)$, denoted as G^k, is a graph over the set of vertices V, in which the vertices $u, v \in V$ are connected by an edge if there exists a path from u to v in G of length at most k. The graph G^2 is called the square of G, while G^3 is called the cube of G.

Hamiltonian cycles in powers of graphs have received considerable attention in the literature. The classical result in this area, by Fleischner [3], is that the square of every 2-connected graph has a Hamiltonian cycle, see also the alternative proofs [2, 4, 12]. Afterwards Hendry & Vogler [7] proved that every connected $S(K_{1,3})$-free graph has a Hamiltonian square (where $S(K_{1,3})$ is the subdivision

⋆ The author is supported by grant no. N206 566740 of the National Science Centre.

graph of $K_{1,3}$), and later Abderrezzak et al. [1] proved the same result for graphs satisfying a weaker condition. As for the cubes of graphs, Karaganis [8] and Sekanina [10] proved that G^3 is Hamiltonian if and only if G is connected. Afterwards, Lin & Skiena [9] gave a linear time algorithm finding a Hamiltonian cycle in the cube of a graph. The analysis of Hamiltonian properties of squares of graphs was also extended to powers of infinite graphs (locally finite graphs), see [5, 10, 11].

In this paper we consider Hamiltonian properties of powers of trees. Harary & Schwenk [6] proved that the square of a tree T has a Hamiltonian cycle if and only if T is a caterpillar, i.e., it is a single path with several leafs connected to it. On the other hand, the k-th power of every tree is Hamiltonian for any $k \geq 3$ [9, 10]. However, there was no previous characterization of trees with squares containing Hamiltonian paths. We introduce a class of trees T, called here (u, v)-horsetails, such that T^2 contains a Hamiltonian path connecting a given pair of nodes u, v if and only if T is a (u, v)-horsetail. We provide linear time algorithms finding Hamiltonian paths in squares of trees.

2 Caterpillars and horsetails

We say that P is a *k-path* in G if P is a path in G^k, similarly we define a *k-cycle*. Additionally define a *kH-path* and a *kH-cycle* as a Hamiltonian path and a Hamiltonian cycle in G^k respectively. If G^k contains a kH-cycle (a kH-path respectively) we call G *k-Hamiltonian* (*k-traceable* respectively).

A tree T is called a *caterpillar* if the subtree of T obtained by removing all the leafs of T is a path B (possibly an empty path). A caterpillar is called non-trivial if it contains at least one edge. We call the path $B = (v_1, \ldots, v_l)$ the *spine* of the caterpillar T. By $Leafs(v_i)$ we denote the set of leafs connected to the node v_i. A tree has a 2H-cycle if and only if it is a caterpillar [6], see Fig. 5.

We proceed to the classification of 2-traceable trees. Let T be a tree and let $u, v \in V$ be two of its nodes, $u \neq v$. Let

$$P = (u = u_1, u_2, \ldots, u_{k-1}, u_k = v)$$

be the path in T connecting u and v. We call the nodes in P the *main* nodes, and the neighbors of the nodes in P from the set $V \setminus P$ are called *secondary* nodes. By $Layer(u_i)$, $i = 1, \ldots, k$, we denote the connected component containing u_i in $T \setminus (P \setminus \{u_i\})$.

Each main node u_i for which $Layer(u_i)$ is a caterpillar can be classified as:

type A: at most one component of $Layer(u_i) \setminus \{u_i\}$ is a non-trivial caterpillar
type B: exactly two components of $Layer(u_i) \setminus \{u_i\}$ are non-trivial caterpillars
free node: $Layer(u_i) \setminus \{u_i\}$ is empty.

In other words, u_i is a type A node if u_i is an endpoint of the spine of the caterpillar $Layer(u_i)$ or a leaf adjacent to the endpoint of the spine. The node u_i is a type B node if u_i is an inner node of the spine of the caterpillar $Layer(u_i)$. Also note that every free node is a type A node.

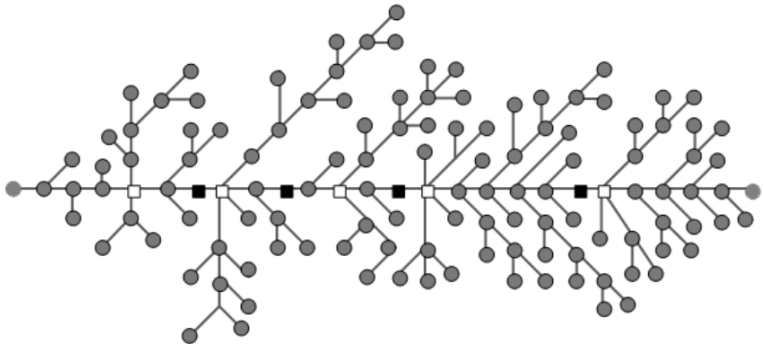

Fig. 1. An example (u, v)-horsetail. White squares represent type B nodes, black squares represent free nodes. The nodes u and v are the leftmost and rightmost nodes on the horizontal path (the main path), they are also free nodes (in this example, in general case they can be non-free).

By $u_{i_1}, u_{i_2}, \ldots, u_{i_m}$, $1 \leq i_1 < i_2 < \ldots < i_m \leq k$, we denote all the type B nodes in T, additionally define $i_0 = 0$ and $i_{m+1} = k + 1$. The tree T is called a (u, v)-*horsetail* if

($\star$) every main node u_i is a type A or a type B node, and
($\star\star$) for every $j = 0, \ldots, m$, at least one of the nodes $u_{i_j+1}, u_{i_j+2}, \ldots, u_{i_{j+1}-1}$ is a free node.

A tree is a *horsetail* if it is a (u, v)-horsetail for some u, v.

The condition ($\star\star$) can also be formulated as follows:

- between any two type B nodes in P there is at least one free node, and
- there is at least one free node located before any type B node in P, and
- there is at least one free node located after any type B node in P, and
- there is at least one free node in P.

In particular, the nodes u_1 and u_k must be type A nodes. See Fig. 1 and 2 for examples of horsetails and trees which are not horsetails.

In the next section we show that T contains a 2H-path connecting the nodes u and v if and only if T is a (u, v)-horsetail, see also Fig. 3. We present a linear time algorithm finding a 2H-path from u to v in a (u, v)-horsetail. Afterwards, in Section 4 we provide a linear preprocessing time algorithm which enables constant time queries for existence of a 2H-path from u to v. We also obtain a linear time test if T is 2-traceable.

3 2H-paths in (u, v)-horsetails

In this section we describe a linear time algorithm finding a 2H-path from u to v in a (u, v)-horsetail. We then show that a tree contains a 2H-path from u to v if and only if it is a (u, v)-horsetail.

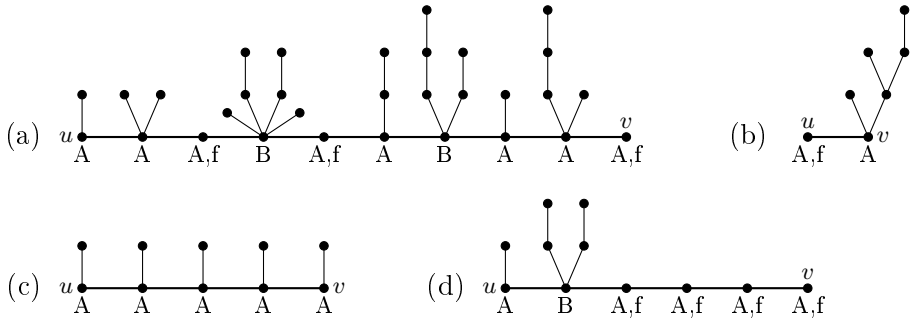

Fig. 2. (a) and (b): The main paths (P) of example horsetails; A, B and f denote a type A node, type B node and a free node respectively. (c) and (d): The main paths of example trees which are not (u, v)-horsetails.

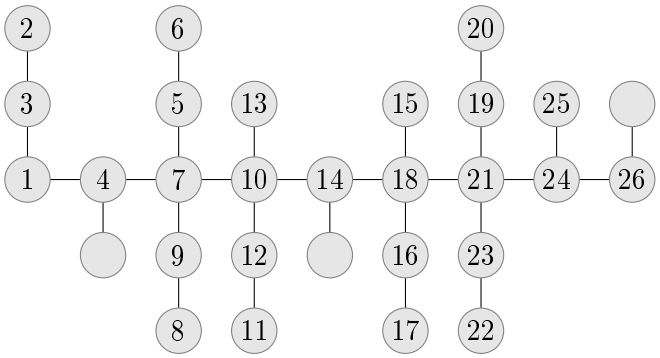

Fig. 3. The tree without the unlabeled nodes is a $(1, 26)$-horsetail having a 2H-path $1, 2, 3, \ldots, 26$. If any of the unlabeled nodes is present in the tree, the tree is not a $(1, 26)$-horsetail and does not contain any 2H-path from 1 to 26.

For a 2-path S, by $\mathit{first}(S)$ and $\mathit{last}(S)$ we denote the first and the last node in S. Let S be a 2H-path from u to v in a (u, v)-horsetail T. We say that S is a *layered* 2H-path if the nodes of T are visited in S in the order of the subtrees $\mathit{Layer}(u_i)$, i.e., first all the nodes of $\mathit{Layer}(u_1)$ are visited in some order, then all the nodes of $\mathit{Layer}(u_2)$ etc. Then we can divide S into parts $(S_1, S_2, \ldots, S_k)$, such that every S_i contains only nodes from $\mathit{Layer}(u_i)$. Note that, for any i, each of the nodes $\mathit{first}(S_i)$, $\mathit{last}(S_i)$ is either a main node or a secondary node.

The following algorithm *2H-path-horsetail* finds a layered 2H-path in a (u, v)-horsetail T, see Fig. 4. Informally, the algorithm works as follows.

In each layer $\mathit{Layer}(u_i)$ a 2H-path S_i is found, using an auxiliary routine *2H-path-caterpillar*$(\mathit{Layer}(u_i), \mathit{first}, \mathit{last})$, and appended to the constructed 2H-path S. The algorithm constructs the 2H-path in a greedy manner: for each i,

$last(S_i) = u_i$, if only this is possible to achieve. If so, we say that after the i-th layer the algorithm is in the *main* phase, otherwise it is in the *secondary* phase.

In the algorithm we use auxiliary functions:

- *any-secondary-node*($Layer(w)$), which returns an arbitrary secondary node from the layer $Layer(w)$;
- *a-secondary-node-preferably-leaf*($Layer(w)$), which finds a secondary node being a leaf in $Layer(w)$, if such a secondary node exists, and any secondary node from this layer otherwise;
- *another-secondary-node*($Layer(w)$), which returns a secondary node from $Layer(w)$ different from the secondary node returned by the previous *a-secondary-node-preferably-leaf*($Layer(w)$) call.

Algorithm 2H-path-horsetail(T, u, v)

```
1: Compute the main nodes u_1, ..., u_k and the layers Layer(u_1), ..., Layer(u_k);
2: first := u_1;
3: if free(u_1) then last := u_1
4: else last := any-secondary-node(Layer(u_1));
5: S := 2H-path-caterpillar(Layer(u_1), first, last);
6: for i := 2 to k do
7:    w := u_i;
8:    if w is a free node then
9:       first := last := w;
10:   else if last is a secondary node then
11:      first := w;
12:      last := a-secondary-node-preferably-leaf(Layer(w));
13:   else
14:      first := a-secondary-node-preferably-leaf(Layer(w));
15:      last := w;
16:   if 2H-path-caterpillar(Layer(w), first, last)=NONE then
17:      last := another-secondary-node(Layer(w));
18:   S := S+2H-path-caterpillar(Layer(w), first, last);
19: return S;
```

Fig. 4. Finding a 2H-path in a horsetail in linear time.

Let us investigate how the algorithm works for respective types of nodes u_i. We have $first(S_1) = u_1$; the 2-path S_1 ends in a secondary node in $Layer(u_1)$ if only u_1 is not free. Otherwise, obviously, $last(S_1) = u_1$. Hence, after $Layer(u_1)$ the algorithm is in the main phase only if u_1 is free.

Afterwards, whenever u_i is a *free* node, we simply visit it, with $first(S_i) = last(S_i) = u_i$. Hence, after any free node the algorithm switches to the main phase.

Otherwise, if u_i is a *type A* node, we have two cases. If the algorithm was in the main phase then we can choose any secondary node for $\mathit{first}(S_i)$ and finish the 2H-path in $\mathit{Layer}(u_i)$ in $\mathit{last}(S_i) = u_i$ — this is the desired configuration. On the other hand, if $\mathit{last}(S_{i-1})$ is a secondary node, then we must have $\mathit{first}(S_i) = u_i$ and we finish in a secondary node. Thus, visiting a non-free type A node does not alter the phase of the algorithm.

Now let u_i be a *type B* node. Due to the $(\star\star)$ condition of the definition of a horsetail, before entering $\mathit{Layer}(u_i)$ the algorithm is in the main phase. Thus we can choose $\mathit{first}(S_i)$ as a secondary node. There is no 2H-path S_i ending in u_i in this case (the proof follows), therefore $\mathit{last}(S_i)$ is another secondary node in $\mathit{Layer}(u_i)$. Hence, a type B node changes the main phase into the secondary phase.

Finally, if u_k is a free node then $\mathit{first}(S_k) = \mathit{last}(S_k) = u_k$ and this ends the path S. Otherwise, we must have $\mathit{last}(S_k) = u_k$, hence $\mathit{first}(S_k)$ must be a secondary node and after the $(k-1)$-th layer the algorithm must be in the main phase. This is, however, guaranteed by the $(\star\star)$ condition of the definition of a horsetail (see also the alternative formulation of the condition).

Thus we obtain the correctness of the algorithm provided that the requested 2H-paths in the caterpillars $\mathit{Layer}(u_i)$ exist. We conclude the analysis with the following Theorem 1. In its proof we utilize the following auxiliary lemma, we omit its proof in this version of the paper.

Lemma 1. *Every 2H-cycle in a caterpillar with the spine $B = (v_1, \ldots, v_l)$, $l \geq 1$, and the leafs $\mathit{Leafs}(v_1), \ldots, \mathit{Leafs}(v_l)$, has the form*

$$v_1\ P_2\ v_3\ \ldots\ v_{l-1}\ P_l\ \ v_l\ P_{l-1}\ \ldots\ P_3\ v_2\ P_1 \quad \textit{for } 2 \mid l$$
$$v_1\ P_2\ v_3\ \ldots\ P_{l-1}\ v_l\ \ P_l\ v_{l-1}\ \ldots\ P_3\ v_2\ P_1 \quad \textit{for } 2 \nmid l$$

where P_i, for $i = 1, \ldots, l$, is an arbitrary permutation of the set $\mathit{Leafs}(v_i)$.

Fig. 5. To the left: a 2H-cycle $1, 2, \ldots, 8$ in a caterpillar with the spine 1–3–7–5 of even length. To the right: a 2H-cycle $1, 2, \ldots, 10$ in a caterpillar with an odd spine 1–3–9–5–7.

Theorem 1. *Assume that T is a (u, v)-horsetail. Then the algorithm* 2H-path-horsetail*(T, u, v) finds a layered 2H-path from u to v in linear time.*

Proof. It suffices to prove that the 2H-path-caterpillar procedure returns the 2H-paths as requested, depending on the type of the node u_i.

If u_i is a type A node (in particular, if $i = 1$ or $i = k$) then the arguments of the 2H-path-caterpillar call are u_i and a secondary node in $\mathit{Layer}(u_i)$. In this case, u_i is either an endpoint of the spine of the caterpillar $\mathit{Layer}(u_i)$ or a leaf of $\mathit{Layer}(u_i)$ connected to an endpoint of the spine. Let $v_1, \ldots, v_l$ be the spine nodes of $\mathit{Layer}(u_i)$; we assume that $l > 0$, since otherwise the conclusion is trivial. Thus, we are looking for a 2H-path in $\mathit{Layer}(u_i)$ connecting the nodes v_1 and w for some $w \in \mathit{Leafs}(v_1)$, or the nodes v_l and w for some $w \in \mathit{Leafs}(v_l)$. Note that we could also have the pair of nodes v_1 and v_2 or v_{l-1} and v_l here, however these cases are eliminated in the algorithm by choosing *a-secondary-node-preferably-leaf*. Equivalently, we are looking for a 2H-cycle in $\mathit{Layer}(u_i)$ containing an edge between the nodes v_1 and w or v_l and w. By Lemma 1, for any $w \in \mathit{Leafs}(v_1)$ ($w \in \mathit{Leafs}(v_l)$ respectively) there exists a 2H-cycle in $\mathit{Layer}(u_i)$ containing the edge v_1w (v_lw respectively), and this 2H-cycle can be found in linear time w.r.t. the size of $\mathit{Layer}(u_i)$. This concludes that the algorithm works correctly in this case.

Now consider any type B node u_i. We will also use the denotation $v_1, \ldots, v_l$ for the spine nodes of $\mathit{Layer}(u_i)$, we have $l \geq 3$. This time u_i is an inner spine node of the caterpillar $\mathit{Layer}(u_i)$, $u_i = v_j$ for some $1 < j < l$. In the algorithm we first try to find a 2H-path in $\mathit{Layer}(u_i)$ connecting this node with a secondary node w, i.e., one of the neighbours of v_j in $\mathit{Layer}(u_i)$, $w \in \{v_{j-1}, v_{j+1}\} \cup \mathit{Leafs}(v_j)$. Equivalently, we search for a 2H-cycle in $\mathit{Layer}(u_i)$ containing an edge v_jw. By Lemma 1, such a 2H-cycle does not exist. Thus, the condition in line 16 of the algorithm is false and we are now looking for a 2H-path in $\mathit{Layer}(u_i)$ connecting two secondary nodes, which are distance-2 nodes in $\mathit{Layer}(u_i)$. This is equivalent to finding a 2H-cycle containing the corresponding edge of $\mathit{Layer}(u_i)^2$. We need to consider a few cases: the node v_{j-1} and a leaf from $\mathit{Leafs}(v_j)$, a symmetric case: v_{j+1} and a leaf from $\mathit{Leafs}(v_j)$, finally v_{j-1} and v_{j+1}. Note that the last case (v_{j-1} and v_{j+1}) is valid only if $\mathit{Leafs}(v_j) = \emptyset$, this is due to the choice of *a-secondary-node-preferably-leaf* in line 14. Now it suffices to note that in all the cases the corresponding 2H-cycle exists and can be found efficiently. Indeed, the cycle from Lemma 1 is of the form $\ldots v_{j+1}P_jv_{j-1}\ldots$ where P_j is any permutation of the set $\mathit{Leafs}(v_j)$. This concludes that also type B nodes are processed correctly. □

Our next goal is to show that T has a 2H-path from u to v if and only if T is a (u, v)-horsetail. First, we note that it is sufficient to consider layered 2H-paths in trees. This is stated formally as the following lemma, we omit the proof in this version of the paper.

Lemma 2. *If T has a 2H-path from u to v then T has a layered 2H-path from u to v.*

This fact already implies that each of the components $\mathit{Layer}(u_i)$ is a caterpillar.

Lemma 3. *If T has a layered 2H-path from u to v then each of the components $\mathit{Layer}(u_i)$ is 2-Hamiltonian.*

Proof. Let S be a layered 2H-path from u to v in T. Then the 2-path S_i is a 2H-path in $Layer(u_i)$. Recall that $first(S_i)$, $last(S_i)$ are either main or secondary nodes in $Layer(u_i)$. Every pair of such nodes are adjacent in the square of $Layer(u_i)$, hence using an additional edge S_i can be transformed into a 2H-cycle in $Layer(u_i)$, so $Layer(u_i)$ is 2-Hamiltonian. □

In the following two lemmas we show that every tree having a 2H-path satisfies the conditions $(\star)$ and $(\star\star)$ of a horsetail.

Lemma 4. *If a tree T contains a 2H-path connecting the nodes u and v then T satisfies the condition $(\star)$ of a (u, v)-horsetail.*

Proof. Assume to the contrary that a node u_i is neither a type A nor a type B node. By Lemma 3, the layer $Layer(u_i)$ is a caterpillar. Let $v_1, \ldots, v_l$ be the spine nodes of $Layer(u_i)$. Then u_i corresponds to some node $w \in Leafs(v_j)$, for $1 < j < l$. Hence, v_j is the only secondary node in $Layer(u_i)$. By Lemma 2, S is a layered 2H-path, $S = (S_1, \ldots, S_k)$. The 2-path S_i is a 2H-path in $Layer(u_i)$ connecting the nodes v_j and w, since both $first(S_i)$ and $last(S_i)$ must be main or secondary nodes of $Layer(u_i)$. Thus S_i is also a 2H-cycle in $Layer(u_i)$. However, by Lemma 1, no such 2H-cycle exists in a caterpillar, a contradiction. □

One can prove the necessity of the $(\star\star)$ condition by showing that after visiting a free node u_i any layered 2H-path in T is in the main phase, that otherwise a type A node keeps the phase of the 2H-path unaltered and that a type B node always changes the main phase into the secondary phase, see Fig. 6.

Lemma 5. *If a tree T contains a 2H-path connecting the nodes u and v then T satisfies the condition $(\star\star)$ of a (u, v)-horsetail.*

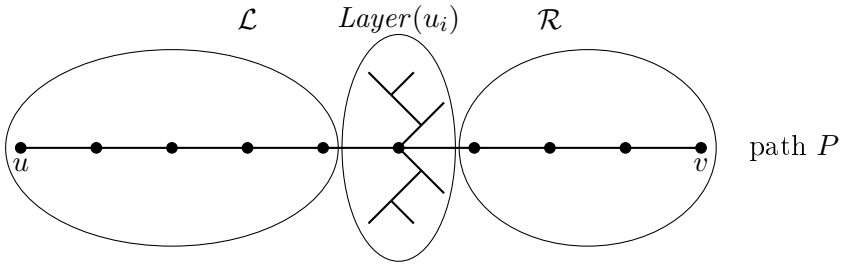

Fig. 6. Let S_L, S_i and S_R be the parts of a layered 2H-path S corresponding to $\mathcal{L}$, $Layer(u_i)$ and $\mathcal{R}$. If u_i is a type B node then $last(S_L) \in P$, $first(S_i), last(S_i) \notin P$, $first(S_R) \in P$.

As a corollary of Lemmas 4 and 5, we obtain the aforementioned result.

Theorem 2. *If a tree T contains a 2H-path connecting the nodes u and v then T is a (u, v)-horsetail.*

4 Efficient algorithm for horstail queries

For a tree $T = (V, E)$, we introduce a *horsetail query* of the form: "*is* T *a* (u, v)*-horsetail for given nodes* $u, v \in V$*?*". In this section we present a linear preprocessing time algorithm which answers horsetail queries for any tree in constant time. We also show how to check if T is 2-traceable (and provide a pair of nodes u, v such that T is a (u, v)-horsetail) in linear time.

We start the analysis with the special case of T being a caterpillar. Answering horsetail queries in this case is easy, we omit the proof of the following simple lemma.

Lemma 6. *Horsetail queries in a caterpillar can be answered in constant time with linear preprocessing time.*

Now we present the algorithm for an arbitrary tree $T = (V, E)$.

1. Let $U = \{w \in V : T \setminus \{w\}$ contains ≥ 3 non-trivial connected components$\}$. Here, again, we call a tree non-trivial if it contains at least one edge.
   (a) If $U = \emptyset$ then T is a caterpillar. Thus T is 2-traceable, and by Lemma 6, horsetail queries in T can be answered in constant time.
   From now on we assume that $U \neq \emptyset$.
2. Let T' be the subtree of T induced by U, i.e., the minimal subtree of T containing all the nodes from the set U.
   (a) If T' is not a path then T is not 2-traceable. Indeed, if T is a (u, v)-horsetail then all the elements of U must lay on the path from u to v.
   From now on we assume that T' is a path $P' = (u'_1, \ldots, u'_m)$, possibly $m = 1$.
3. By L_i, $i = 1, \ldots, m$, we denote the connected component containing u'_i in $T \setminus (P' \setminus \{u'_i\})$. If T is a (u, v)-horsetail then $u \in L_1$ and $v \in L_m$ (or symmetrically).
   (a) Check if every component $L_2, \ldots, L_{m-1}$ satisfies part $(\star)$ of the definition of a horsetail, in particular, if it is a caterpillar (a linear time test). If not then T is not 2-traceable.
4. Let $D_1, \ldots, D_p$ and $F_1, \ldots, F_q$ be the connected components of $L_1 \setminus \{u'_1\}$ and $L_m \setminus \{u'_m\}$ respectively, ordered by the number of nodes (non-increasingly). Note that, by the definition of the set U, each D_i and each F_i is a caterpillar.
   (a) If $m > 1$ and at least one of the subtrees D_4, F_4 exists and is non-trivial then T is not 2-traceable. Indeed, regardless of the choice of u and v, the layer containing the node u'_1 (u'_m respectively) would not be a caterpillar. Similarly, if $m = 1$, D_5 exists and is non-trivial then T is not 2-traceable.
5. Denote by X_1 the set of all the nodes w of non-trivial components D_i such that the path from w to u'_1 contains at least one free node (excluding u'_1). By X_2 we denote the set of all the remaining nodes of non-trivial components D_i. Finally, by X_3 we denote the set of all the nodes of all the trivial components D_i. Analogically, we define the sets Y_1, Y_2 and Y_3, replacing u'_1 with u'_m and D_i with F_i.

(a) Assume that $m > 1$. For every pair of indices $1 \leq a, b \leq 3$, pick any nodes $u \in X_a$ and $v \in Y_b$ (provided that $X_a \neq \emptyset$ and $Y_b \neq \emptyset$) and check if T is a (u, v)-horsetail — it suffices to check here if u'_1 and u'_m satisfy the condition $(\star)$ and if the path from u to v satisfies the condition $(\star\star)$. If so then for every pair of nodes $u \in X_a$ and $v \in Y_b$ the tree T is a (u, v)-horsetail, otherwise T is not a (u, v)-horsetail for any such pair.

(b) Assume that $m = 1$. Similarly as in the step 5(a), we consider any pair of nodes u, v from the sets X_a and X_b. This time, however, we pick any elements from *different* components (i.e., $u \in D_i$, $v \in D_j$ for $i \neq j$). Again we see that the condition $(\star)$ for u'_1 and the condition $(\star\star)$ hold for the chosen nodes u and v if and only if these conditions hold for any pair of nodes from different components from X_a and X_b.

In both cases we obtain a constant number of pairs of sets A_i, B_i such that T is a (u, v)-horsetail if and only if $u \in A_i$ and $v \in B_i$ for some index i, and u, v are in different components D_j in the case $m = 1$.

Clearly, the above algorithm can be implemented in linear time, it suffices to employ depth-first search of selected subtrees of T. We obtain the following result.

Theorem 3. *Horsetail queries in any tree can be answered in constant time with linear preprocessing time. Moreover, testing if a tree is 2-traceable can be done in linear time.*

References

1. M. E. K. Abderrezzak, E. Flandrin, and Z. Ryjáček. Induced $S(K_{1,3})$ and Hamiltonian cycles in the square of a graph. *Discrete Mathematics*, 207(1-3):263–269, 1999.
2. R. Diestel. *Graph Theory (4th ed.)*. Springer-Verlag, Heidelberg, 2010.
3. H. Fleischner. The square of every two-connected graph is Hamiltonian. *J. Combin. Theory (Series B)*, 16:29–34, 1974.
4. A. Georgakopoulos. A short proof of Fleischner's theorem. *Discrete Mathematics*, 309(23-24):6632–6634, 2009.
5. A. Georgakopoulos. Infinite Hamilton cycles in squares of locally finite graphs, Preprint 2006.
6. F. Harary and A. Schwenk. Trees with Hamiltonian square. *Mathematika*, 18:138–140, 1971.
7. G. Hendry and W. Vogler. The square of a $S(K_{1,3})$-free graph is vertex pancyclic. *Journal of Graph Theory*, 9:535–537, 1985.
8. J. J. Karaganis. On the cube of a graph. *Canad. Math. Bull.*, 11:295–296, 1968.
9. Y.-L. Lin and S. Skiena. Algorithms for square roots of graphs. *SIAM J. Discrete Math.*, 8(1):99–118, 1995.
10. M. Sekanina. On an ordering of the set of vertices of a connected graph. Technical Report 412, Publ. Fac. Sci. Univ. Brno, 1960.
11. C. Thomassen. Hamiltonian paths in squares of infinite locally finite blocks. *Annals of Discr. Math.*, 3:269–277, 1978.
12. S. Řiha. A new proof of the theorem by Fleischner. *J. Comb. Theory Ser. B*, 52:117–123, May 1991.