A Linear-Time Kernelization for the Rooted k-Leaf Outbranching Problem

Frank Kammer

Institut für Informatik, Universität Augsburg, 86135 Augsburg, Germany
kammer@informatik.uni-augsburg.de

Abstract. In the ROOTED k-LEAF OUTBRANCHING PROBLEM, a digraph $G = (V, E)$, a vertex r of G, and an integer k are given, and the goal is to find an r-rooted spanning outtree of G with $\geq k$ leaves (a tree with vertex set V, all edges directed away from r, and $\geq k$ leaves). We present a linear-time algorithm to compute a problem kernel with $O(k^6)$ vertices and $O(k^7)$ edges for the ROOTED k-LEAF OUTBRANCHING PROBLEM. By combining the new result with a result of Daligault and Thomassé [IWPEC 2009], a kernel with a quadratic number of vertices and edges can be found on n-vertex m-edge digraphs in time $O(n + m + k^{14})$.

1 Introduction

To solve an NP-hard problem P, a common approach is to consider P as a *parameterized problem* described by a language $L \subseteq \Sigma^* \times \mathbb{N}$. An instance of P is a tuple $(x, k) \in \Sigma^* \times \mathbb{N}$ composed of the problem specification x and a problem-specific *parameter* k. We call (x, k) a *yes instance* if $(x, k) \in L$, and otherwise a *no instance*. The goal is to solve P in $n^c \cdot f(k)$ time, where n is the size of the input, c is a constant, and f is an arbitrary function. There are two research lines; one that tries to make f grow as slowly as possible and one that tries to achieve the smallest possible c; one example of the latter is the linear-time algorithm for TREEWIDTH [5].

In recent years, several polynomial-time algorithms were published that transform an instance (x, k) into a tuple $(x', k') \in \Sigma^* \times \mathbb{N}$ such that $|x'|$ and k' are bounded by a polynomial in k and such that (x, k) is a yes instance if and only if (x', k') is a yes instance. Such an algorithm is called a *kernelization*, and (x', k') is a *kernel* for (x, k). For numerous problems, a race has started to obtain better and better bounds on the size of the kernel and, thus, on the corresponding running times with more and more slowly growing functions f. In contrast, the only linear-time kernelizations known to the author are for VERTEX COVER [7], for (BI)CLUSTER GRAPH EDITING [11] on general undirected graphs, for d-HITTING SET [2] on undirected hypergraphs, and for FEEDBACK VERTEX SET [10] and DOMINATING SET [3,8] on undirected planar graphs.

We present a linear-time algorithm that, given a general digraph G and a parameter $k \in \mathbb{N}$, computes a kernel of (G, k) with $O(k^6)$ vertices and $O(k^7)$ edges for the ROOTED k-LEAF OUTBRANCHING PROBLEM. As usual, an *outtree*

T is a rooted tree in which all edges are directed away from a vertex r called *the root*. Other vertices with only one incident edge are called *leaves*. A vertex u is an ancestor (descendant) of a vertex v in T if there is a directed path from u to v (from v to u) in T. An ancestor or descendant u of a vertex v is called *proper* if $u \neq v$. A *child* (*parent*) of a vertex v is a proper descendant (ancestor) of v that is adjacent to v.

Definition 1. *A* k-leaf outtree (r-rooted outtree) *is an outtree with at least* $k \in \mathbb{N}$ *leaves* (*with root* r). *An* outbranching *of a digraph* $G=(V, E)$ *is an outtree* $T = (V, F)$ *with* $F \subseteq E$. *In the* (Rooted) k-Leaf Outbranching Problem, *we are given a digraph* $G = (V, E)$, $k \in \mathbb{N}$ *(and* $r \in V$*), and the goal is to find an (*r*-rooted)* k*-leaf* outbranching *of* G.

In 2009 Fernau et al. [4] showed that an $O(k^3)$-sized kernel can be found for the Rooted k-Leaf Outbranching Problem in polynomial time, whereas this is not true for the related non-rooted version unless $PH = \Sigma_p^3$. In the same year, Daligault and Thomassé [6] described an algorithm to compute a kernel of size $O(k^2)$ for the Rooted k-Leaf Outbranching Problem. An efficient implementation of the algorithm runs in quadratic time. By running the algorithm of Daligault and Thomassé on the output of the algorithm presented in this paper, we can construct a kernel of size $O(k^2)$ on a digraph (V, E) in time $O(|V| + |E| + k^{14})$. If there is a polynomial-time algorithm (including algorithms still to be discovered) that computes a kernel of a certain size for this problem, we get a kernel of the same size in $O(|V| + |E| + k^{O(1)})$ time with the new algorithm.

2 Reduction Rules

A *reduction rule* for a parameterized problem P described by a language L is a polynomial-time algorithm that, *applied* to an instance (x, k) of P, computes an instance (x', k') of P with $(x, k) \in L \Leftrightarrow (x', k') \in L$. Our algorithm is based on one new and several well-known reduction rules [4,6], which we apply in a certain order. Assume that we are given an instance of the parameterized Rooted k-Leaf Outbranching Problem, i.e., a digraph $G = (V, E)$, $r \in V$, and $k \geq 2$. For all $v \in V$, let $N^-(v) = \{u \mid (u, v) \in E\}$ and $N^+(v) = \{u \mid (v, u) \in E\}$. A set $S \subseteq V$ is called a *separator for a set* $U \subseteq V \setminus S$ if all directed paths from r to a vertex in U contain a vertex in S. Following usual terminology, if $S = \{v\}$ is a separator for $U = \{u\}$, we say that v is a *dominator of* u and that u is *dominated* by v. We call a vertex a *dominator* if it dominates at least one other vertex. An edge (u, v) is *useless* if v is a dominator for u, and *useful* otherwise. A list L of vertices $(v_1, \ldots, v_t)$ ($t \in \mathbb{N}$) is a *bipath* of *length* $t - 1$ in a digraph G' if, for all $i = 2, \ldots, t-1$, v_i is incident to (v_{i-1}, v_i), (v_i, v_{i-1}), (v_i, v_{i+1}), and (v_{i+1}, v_i), but to no other edge in G'. We now define our reduction rules.

Rule 1 (unreachable rule (Rule 1 in [4] and 0 in [6])). *If a vertex is not reachable from* r*, reduce the given instance to a trivial no instance.*

Rule 2 (useless-edge rule at (u, v) (Rule 2 in [4])). *If the edge (u, v) is useless, remove it.*

Rule 3 (separator rule at (u, w) (Rule 4 in [4])). *Remove the edge (u, w) if there is a separator $S \subseteq V \setminus \{r, u\}$ for $\{u\}$ of size at most 2 such that an edge (v, w) exists for all $v \in S$.*

Rule 4. (dominator rule at v (Rule 1 in [6] combined with Rule 2 in [4])) *If v is a dominator, then remove v with all its incident edges and add all edges from each vertex in $N^{-}(v)$ to each vertex in $N^{+}(v)$ except those that would be useless, self-loops or copies of existing edges.*

Rule 5 (bipath rule (Rule 2 in [6])). *If $P = (v_1, \ldots, v_t)$ is a bipath of length ≥ 5, then replace P by the bipath (v_1, v_2, v_{t-1}, v_t).*

Rule 6 (shortcut rule at v (new)). *If there are pairwise distinct vertices u, x, y with $N^{-}(u) = \{v\}$ and edges (y, v) and (v, x), then add a new edge (y, x) if it does not already exist.*

The correctness of Rules 1-5 follows from [4,6]. Consider the situation described in Rule 6. A k-leaf outbranching in the original digraph is a k-leaf outbranching in the modified digraph. For the reverse direction, take an outbranching T that uses (y, x). By modifying T, we obtain an outbranching with at least the same number of leaves, but without the edge (y, x) as follows: If v is a descendant of y in T, replace the edge from the parent of v in T to v by the edge (y, v). At this point, v cannot be a descendant of x. Now replace (y, x) by (v, x). Before and after the replacement, v has u as a child. Thus the modifications transform an outtree into an outtree with at least the same number of leaves. Rule 6 is correct.

The separator rule is applied exhaustively by Fernau et al. [4], so that the running time can be bounded only by a polynomial. In contrast, we use the separator rule only if we already know the separator S. Moreover, a single application of the dominator rule, which is used by Daligault and Thomassé [6], can add a quadratic number of edges and, hence, by itself incurs a quadratic running time. Informally speaking, the new shortcut rule allows us to do the modifications of the dominator rule one by one.

If the reduction rules are applied repeatedly in an interspersed fashion, one application of a reduction rule usually scans the whole graph and therefore takes linear time. An additional idea of our algorithm is to structure the application of the reduction rules by means of a breadth-first search (BFS). In contrast to [6,4], our algorithm uses a *dominator tree* [1,9] to apply the useless-edge rule in constant time per edge.

3 Definitions and the Algorithm

Let $G = (V, E)$ be a digraph and let $r \in V$. For an r-rooted outtree $T = (V, F)$ with $F \subseteq E$, we *classify* the edges (u, v) of G with respect to T as follows: each edge in T is a *tree edge*; a non-tree edge (u, v) is a *forward edge* (*back edge*) if T contains a directed path from u to v (from v to u); the remaining edges of G are *cross edges*. When classifying an edge of G w.r.t. T, we also speak of a tree, forward, back, or cross edge of (G, T). For an edge (u, v), we call u and v the *tail* and the *head of* (u, v), respectively. We also say that (u, v) is an *incoming edge of* v and an *outgoing edge of* u. *To contract* an edge (u, v) means to replace u and v by a new vertex w, then to replace the endpoints u and v of all edges by w, and finally to remove self loops and multiple edges.

We also need some non-standard definitions. Call a vertex u of T a *branching vertex* if u is the root of T or has ≥ 2 children in T. A *treepath* of (G, T) is a directed path in T without a branching vertex. For a vertex v belonging to a treepath P of (G, T), edges of the forms (u, v) and (v, w) are called *entering edges of* P and *exiting edges of* P, respectively, if neither they nor their reverses are tree edges and u and w do not belong to P. A maximal (nonextensible) subpath Q of a maximal treepath P such that none of the vertices of Q is the head of an entering edge of P is called an *isolated treepath* of (G, T). If (G, T) or P are clear from the context, we may omit these terms. When we later apply the shortcut rule, we want to add edges connecting so-called essential vertices of the same isolated treepath. A vertex v is *essential* if it is part of an isolated treepath and not dominated by any vertex in the same isolated treepath. Unless stated otherwise, here and below "domination" is meant with respect to the whole graph G. Note that an essential vertex has an incoming back edge unless it is the first vertex of an isolated treepath or the head of a forward edge. The *attachment* of an essential vertex v of an isolated treepath Q is the (possible empty) set consisting of all vertices of Q dominated by v.

Our algorithm is shown in pseudocode below—the concepts of "jumping back edges" and "strongly isolated treepaths" are defined subsequently. Suppose that we are given an input consisting of a digraph G, a prescribed root r and an integer k. An important idea of the algorithm is to apply a sequence of reduction rules to G and corresponding modifications to an easy-to-compute r-rooted outbranching T of G to arrive at a pair $(\tilde{G}, \tilde{T})$ for which, roughly speaking, the number of non-tree edges is polynomial in k. With such a pair $(\tilde{G}, \tilde{T})$, the set V' of vertices incident on non-tree edges is of size $k^{O(1)}$. Assume that $\tilde{T}$ has fewer than k leaves (otherwise, we can return a trivial yes instance). Then the set V'' of branching vertices is smaller than k, and $\tilde{G}$ has fewer than $2k$ maximal treepaths. The vertices outside of $V' \cup V''$ induce $k^{O(1)}$ vertex-disjoint paths in $\tilde{G}$. Let G' be the graph obtained from $\tilde{G}$ by applying the bipath-rule to each of these paths to shrink it to constant length. Since G' has $k^{O(1)}$ vertices and we apply only reduction rules, (G', k) is a kernel.

Initially, we take T to be an arbitrary outbranching computed by a BFS in Step 1 and 2. Thus, (G, T) has no forward edges. As we show later in Lemma 3, this property is maintained during the whole algorithm.

For the next steps, consider an isolated treepath $Q = (q_1, q_2, q_3, \ldots)$ as the example in Fig. 1. Let w be an essential vertex of Q with a non-empty attachment S. By Lemma 4, each $v \in S$ can have only a tree edge and useless back edges

Algorithm 1. A kernel for the r-Rooted k-Leaf Outbranching Problem

```
Input: A digraph $G=(V,E)$, $r \in V$, and $k \geq 2$.  Output: A kernel of size $O(k^7)$.
1.) if some vertex is not reachable from $r$, then return a trivial no instance.
2.) Compute an outbranching $T$ with root $r$ by a BFS.
3.) Remove all useless back edges of $(G,T)$ by using a dominator tree—see Sect. 4.
4.) Traverse $T$ bottom-up, and for each vertex $u$ in $G$ with parent $v$ in $T$ do
     if  $(N^-(u) = \{v\}$ /* $v$ is a dominator */ ) and $(|N^-(v)| = 1)$
               and (neither $v$ nor its parent in $T$ is a branching vertex) then
        Apply the dominator rule at $v$ both to $G$ and to $T$.
5.) for each strongly isolated treepath $R$ of $(G,T)$ do for each vertex $v$ of $R$
     that dominates its child and has a non-exiting back edge $(v,x)$ do
     Let $(y,v)$ be a back edge. Apply the shortcut rule at $v$ to add $(y,x)$.
6.) for each strongly isolated treepath $R=(u_1,\ldots,u_t)$ of $(G,T)$ do
     for each vertex $w$ not in $R$ with $\ell \geq 3$ edges of the form (vertex in R$, w$) do
         Choose $\ell$ maximal and $1<\sigma_1<\ldots<\sigma_\ell<t$ s.t. $(u_{\sigma_1},w),\ldots,(u_{\sigma_\ell},w) \in E$.
         $x := \ell$; if $u_{\sigma_{\ell-1}}$ is a dominator of $u_{\sigma_\ell}$, then $x := \ell - 1$
         Apply the separator rule with $S = \{u_{\sigma_1}, u_{\sigma_x}\}$ at each edge
         $e \in \{(u_{\sigma_2},w),\ldots,(u_{\sigma_\ell},w)\} \setminus \{(u_{\sigma_x},w)\}$ to remove $e$.
7.) if ($T$ has $\geq k$ leaves)
    || (a maximal treepath of $(G,T)$ has $\geq 2k$ different heads of entering edges)
    || (a vertex of a maximal treepath $P$ is the tail of $\geq k$ back edges not exiting $P$)
    || (an isolated treepath of $(G,T)$ has $\geq 4k^2$ heads of jumping back edges)
    then return a trivial yes instance.
8.) for each strongly isolated treepath $R=(u_1,\ldots,u_t)$ of $(G,T)$ do
     for each maximal subpath $R'$ of $R$ consisting exclusively of vertices
       such that each vertex is not the tail of an exiting or jumping back edge do
         Apply the dominator rule to each vertex in $R'$ dominating its child. Let
         $R''$ be the path obtained. Apply the bipath rule to shrink $R''$ to length
         $O(1)$.
9.) Let $G'$ be the digraph obtained in Step 8. return $(G',k)$.
```

as incoming edges. Since there are no useless back edges after their deletion in Step 3 (Lemma 5), each vertex in S has only one incoming edge, which is a tree edge. If we consider the subpath Q' of Q induced by $S \cup \{w\}$, each vertex with a successor is a dominator. In Step 4, the if conditions are satisfied for each dominator v in Q', and we can remove v by applying the dominator rule at v. Afterwards the attachment of w is of size one. To sum up, each isolated treepath consists of a first vertex, essential vertices, and attachments of size one after Step 4.

We next consider subpaths of isolated treepaths. If u and v are vertices of the same isolated treepath Q and if the path from v to u in Q contains ≥ 2 essential vertices $w_1 \neq v$ and $w_2 \neq v$, we call a back edge (u,v) (Q-)*jumping*. Intuitively, such an edge jumps over a vertex that can be made a leaf if we traverse Q backwards—in Fig. 2, a z-rooted outbranching with ≥ 2 leaves needs a jumping back edge. A maximal subpath R of an isolated treepath $Q = (q_1, q_2, q_3, \ldots)$

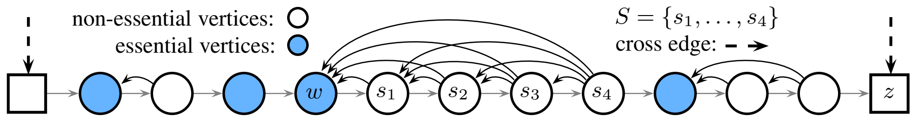

Fig. 1. An isolated treepath consisting of the round vertices with tree edges shown as straight arrows. The essential vertices define which back edges are useless. All possible useless edges are shown as curved arrows. In addition, each essential vertex is head of a useful but non-displayed back edge whose tail is for example z.

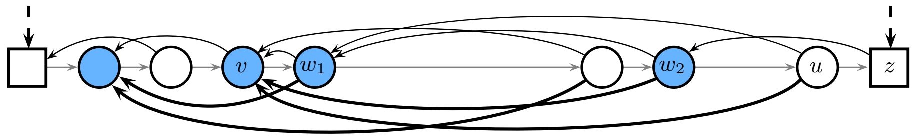

Fig. 2. Jumping (bold) and non-jumping back edges (thin)

without q_1 and without heads of Q-jumping back edges is called a *strongly isolated treepath of* (G, T)—by excluding q_1, R has only essential vertices and attachments of size one.

In Step 6, we need the following property for each strongly isolated treepath R: If a vertex v in R is dominating its child, all back edges $e = (v, x)$ are exiting R. To ensure that the property holds, in Step 5 the algorithm chooses a vertex y such that (y, v) is a back edge—such an edge must exist since v is essential and not the first vertex of an isolated treepath—and adds a jumping back edge $e' = (y, x)$ so that e becomes exiting (for more details see Lemma 6 and its proof). Note that e' is a back edge since (y, v) and e are back edges. In Step 6, for each strongly isolated treepath $R = (u_1, \ldots, u_t)$ and each vertex w not in R, we remove all except 2 (cross or jumping back) edges of the form (u_i, w) $(1 \leq i \leq t)$ by applying the separator rule. Lemma 7a shows that the set S chosen by the algorithm is indeed a separator. In Step 7, we check several conditions that reveal an obvious yes instance. The correctness is shown in Section 6. In Step 8, we iterate over all strongly isolated treepaths $R = (u_1, \ldots, u_t)$ and all maximal subpaths R' of $(u_2, \ldots, u_t)$ without tails of exiting and jumping back edges. By Lemma 7b, the application of the dominator rule at each vertex with a non-empty attachment turns R' into a bipath, which is then shrunk by the bipath rule.

4 Running Time

A outtree T and the classification of the edges in Step 2 can be computed in linear time, i.e., linear in the number of vertices and edges. Recall that an edge (u, v) is useless if and only if v dominates u. Step 3 can also be executed in linear time since a dominator tree allows queries of the form "Is u a dominator of v?" to be answered in constant time and can be constructed in linear time [1,9]. In Step 4, the traversal can be done in linear time since the if condition of $(|N^-(v)| = 1)$

guarantees that we only have to replace each outgoing edge of v by an outgoing edge of its parent. In Step 5, by two scans over all edges, we can split all treepaths in isolated and subsequently in strongly isolated treepath. Using a dominator tree we can find each v that dominates its child. By iterating over all outgoing edges of v, we compute all non-exiting back edges. For each such edge, we add one new edge to G. Thus, the running time of this step is linear. Step 6 can be implemented in linear time by using radix sort to find the exiting edges of each strongly isolated treepath. We so know all vertices w for which the inner for loop is executed. In Step 8, the time to process each path R' is linear in the number of vertices of R' by Lemma 7b.

5 Properties of the Computation

We start with an auxiliary lemma. Many proofs are skipped.

Lemma 2. *Let $\tilde{G}$ be a digraph with an r-rooted outbranching $\tilde{T}$ such that $(\tilde{G}, \tilde{T})$ has no forward edges. Let $e = (p(v), v)$ be a tree edge with $N^-(v) = \{p(v)\}$ and assume that neither $p(v)$ nor v is a branching vertex. Applying the cutvertex rule at v in $\tilde{G}$ and in $\tilde{T}$ does not change the classification (back edge, etc.) of any edge $e' \notin \{(p(v), v), (v, p(v))\}$. Moreover, a useful back edge remains useful after the application of the cutvertex rule.*

Lemma 3. *(G, T) has no forward edges throughout Steps 1 to 7.*

Lemma 4. *Let G and T as in the algorithm after Step 1. Let w be an essential vertex whose attachment contains a vertex v. Then v has as incoming edges only one tree edge and useless back edges.*

Proof. Let (x, v) be an edge of G. By Lemma 3, (x, v) is not a forward edge. Assume that (x, v) is a cross edge. Since w is not a branching vertex, the path P from the root r to x in T cannot contain w. Taking P and the edge (x, v) we obtain a path Q from r to v that avoids w.

Assume that (x, v) is a useful back edge. Let P be a path from r to x avoiding v with a minimal number of cross and back edges. Note that w is not a vertex of P since otherwise, P would use a cross or back edge (w, y) to avoid v—y is consequently not a descendant of v—and the path from r to y consisting of tree edges combined with the subpath of P from y to x avoids v and has fewer cross and back edges than P. Thus, P and (x, v) defines a path Q from r to v that avoids w.

In both cases, v is not part of the attachment of w—contradiction. □

Lemma 5. *(G, T) has no useless back edges throughout Steps 4 to 7.*

Lemma 6. *After Step 5, if a vertex v of a strongly isolated treepath Q dominates its child, each back edge $e = (v, x)$ is exiting Q.*

Proof. The attachment of v is $\{\text{child of } v\}$. Since a vertex of a strongly isolated treepath is not the first vertex of an isolated treepath, recall that v is not dominated by its parent after Step 4. Thus, v has an incoming non-tree edge $e' = (y, v)$, which must be a non-jumping back edge since v is a vertex of a strongly isolated treepath.

Assume that e is not exiting. Then, e is a non-jumping back edge. See Fig. 3. In Step 5, all conditions regarding v are satisfied and the shortcut rule applied to v adds an edge (y, x). Thus, Q is no strongly isolated treepaths after Step 5. Contradiction, and e must be exiting. □

Lemma 7. *Let $R = (u_1, \ldots, u_t)$ be a strongly isolated treepath.*

a) *In Step 6, $S = \{u_{\sigma_1}, u_{\sigma_\ell}\}$ is a separator for $\{u_{\sigma_2}, \ldots, u_{\sigma_{\ell-1}}\}$ except if $u_{\sigma_{\ell-1}}$ dominates u_{σ_ℓ}, in which case $S = \{u_{\sigma_1}, u_{\sigma_{\ell-1}}\}$ is a separator for $\{u_{\sigma_2}, \ldots, u_{\sigma_{\ell-2}}, u_{\sigma_\ell}\}$.*

b) *If a subpath R' of R consists exclusively of vertices such that each vertex is not the tail of an exiting or jumping back edge, the application of the dominator rule to each vertex of R' that dominates its child turns R' into a bipath and can be done in time $O(|\text{vertices of } R'|)$.*

Proof. After Step 5, the distribution of the essential vertices of R fixes the back edges with both endpoints in R: Each essential vertex v of R is the head of a back edge e. Since e can neither be jumping nor useless and because of Lemma 6, the tail of e is the first essential vertex w after v in R if w does not dominate its child, and the only vertex in the attachment of w otherwise. Fig. 4 shows examples of all cases. To sum up, if repeated visits are forbidden, there is a unique way to visit the vertices of R if we start with the first (last) vertex of R. In the forward direction we can use only the tree edges. In the backward direction, we must use a tree edge at each vertex with a non-empty attachment, and a back edge otherwise.

a) Note that each path P from the root r to a vertex of Umust visit one endpoint $\tilde{u}$ of R. If we now consider the subpath of P starting in $\tilde{u}$, we can observe that P has to visit a vertex of S before it can reach $U = \{u_{\sigma_1}, \ldots, u_{\sigma_\ell}\} \setminus S$. In each of the two cases, S is a separator for U.

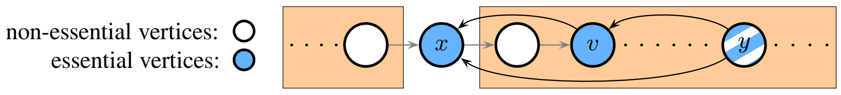

Fig. 3. A strongly isolated treepath (consisting of all vertices above) is split into two parts by Step 5, and (y, x) and (v, x) are exiting

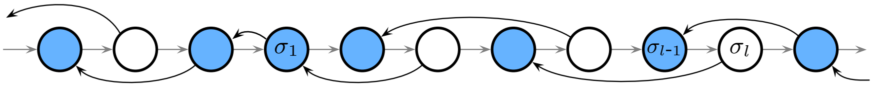

Fig. 4. A strongly isolated treepath after Step 5. Exiting edges are not shown.

b) Let us group the vertices of R'. Each vertex dominated by its parent builds a group with its parent. Every other vertex forms a group by itself. Then there is exactly one back edge from each group to the previous group. The application of the dominator rule contracts each group, which turns R' into a bipath, and this can be done in $O(1)$ time per group. □

6 Correctness of Step 7

If the first condition of Step 7 holds, the answer of the algorithm is correct. Thus, for the remaining analysis, we can assume that T has fewer than k branching vertices, and the number of maximal treepaths of (G,T) is less than $2k$.

The correctness of the second and the third condition of Step 7 is shown by Lemmata 8 and 9, respectively. The requirements of the lemmata are satisfied by Lemmata 3 and 5. Since we want to use Lemma 8 again in the proof of Lemma 10, we allow a certain kind of forward edges.

Lemma 8. *Let $\tilde{G}$ be a digraph, and let $\tilde{T}$ be an outbranching of $\tilde{G}$ with root r such that every forward edge of $(\tilde{G},\tilde{T})$ has tail r. If no back edge of $(\tilde{G},\tilde{T})$ is useless and if a treepath of $(\tilde{G},\tilde{T})$ has $\geq 2k$ different heads of entering edges, then $\tilde{G}$ has a k-leaf outbranching.*

Lemma 9. *Let $\tilde{G}$ be a digraph with an outbranching $\tilde{T}$ such that $(\tilde{G},\tilde{T})$ has no forward edges. If a vertex u of a maximal treepath P is the tail of $\geq k$ useful back edges that do not exit P, $\tilde{G}$ has a k-leaf outbranching.*

Assume that in Step 7 an isolated treepath P of (G,T) has $\geq 4k^2$ heads of jumping back edges. Let D be the set of essential vertices of (G,T) that have an attachment. Recall that the following property holds: each vertex v in D with its parent $p(v)$ in the same isolated treepath is not dominated by $p(v)$ after Step 4.

For the analysis, consider the digraph $\tilde{G}$, the outtree $\tilde{T}$, and the directed path $\tilde{P}$ obtained from G, T, and P, respectively, by contracting each vertex $v \in D$ with the only vertex in the attachment of v. By Lemmata 3 and 5, (G,T) has neither forward nor useless back edges. Lemma 2 shows that this property is maintained in $(\tilde{G},\tilde{T})$. Moreover, each jumping back edge (u,v) of (G,T) is a jumping back edge of $(\tilde{G},\tilde{T})$ since the number of essential vertices on the v-u-path is not changed by the contract operation. Also the number of heads of jumping back edges do not change. Let $v_1,\ldots,v_t$ be the vertices of $\tilde{P}$, and for each vertex v_i $(2 \leq i \leq t)$, denote by $p_{\tilde{T}}(v_i)$ its parent in $\tilde{T}$. By the property mentioned in the last paragraph, each vertex v_i $(2 \leq i \leq t)$ is not dominated by $p_{\tilde{T}}(v_i)$ in $\tilde{G}$, i.e., v_i is the head of a non-tree edge e. Since v_i cannot be the head of a cross edge and since no edges are useless, e is a useful back edge. By Lemma 10, $\tilde{G}$ has an outbranching with at least k leaves. It is easy to see by "undoing" the contraction from above that the outbranching for $\tilde{G}$ can be transformed into an outbranching for G without decreasing the number of leaves. Thus, the last condition of Step 7 is correct.

Lemma 10. *Let $\tilde{G}$ be a digraph with an outbranching $\tilde{T}$ such that $(\tilde{G}, \tilde{T})$ has no forward and no useless back edges. Let r be the root of $\tilde{T}$, and let $P = (v_1, \ldots, v_t)$ $(t \in \mathbb{N})$ be an isolated treepathsuch that each vertex v_i $(i \geq 2)$ is the tail of a useful back edge. If P contains $\geq 4k^2$ heads of jumping back edges, $\tilde{G}$ has an r-rooted outbranching with $\geq k$ leaves.*

7 Kernel Size

Next, we want to analyze the size of the kernel returned in Step 9. For this purpose, it is interesting to bound the number of maximal subpaths of a strongly isolated treepath consisting exclusively of vertices that are not a tail of an exiting or jumping back edge.

Lemma 11. *After Step 7, the number of heads of entering edges of a maximal treepath of (G, T) is less than $2k$ and the number of exiting edges with a tail in a fixed isolated treepath of (G, T) is at most $44k^4$.*

Proof. T has fewer than k branching vertices by the first condition of Step 7. Thus, (G, T) has at most $2k$ maximal treepaths. Moreover, each treepath of (G, T) has fewer than $2k$ heads of entering edges due to the second condition of Step 7. Let z be the number of exiting edges with a tail in a fixed strongly isolated treepath R that is a subpath of a maximal treepath P. Because of Step 6, z is bounded by twice the number of heads of entering edges of the other maximal treepaths $P' \neq P$ plus the number of branching vertices since a head of an exiting edge of R is either a branching vertex or the head of an entering edge of some other maximal treepath $P' \neq P$. By the pigeon-hole principle R has fewer than $2(k + 2k \cdot 2k) \leq 10k^2$ tails of exiting edges. By the fourth condition of Step 7, each isolated treepath Q has fewer than $4k^2$ heads of jumping back edges, i.e., Q can be divided into at most $4k^2$ strongly isolated treepaths. Since each vertex of Q either belongs to a strongly isolated treepath or is one of the at most $4k^2$ remaining vertices, there are at most $4k^2 \cdot 10k^2 + 4k^2 = 44k^4$ exiting edges with a tail in Q. $\square$

Lemma 12. *After Step 7, for an isolated treepath P, the number of heads and tails of jumping back edges is less than $4k^2$ and $32k^4$, respectively.*

Proof. By the fourth condition of Step 7, P can be divided into at most $4k^2$ strongly isolated treepaths. By Step 6, each vertex can be the head of two jumping back edges with a tail in the same strongly isolated treepath P' subpath of P. In total, each vertex can be the head of at most $8k^2$ jumping back edges. Since P has at most $4k^2$ vertices being a head of jumping back edge, P has at most $32k^4$ tails of jumping back edges. $\square$

By the last two lemmata, the number of isolated treepaths is $k \cdot 2k = O(k^2)$ in total, and each such treepath has $44k^4 + 4k^2 + 32k^4 = O(k^4)$ tails of exiting edges and endpoints of jumping back edges. Thus, we have $O(k^6)$ directed paths R' in Step 8, which are shrunk to length $O(1)$. Since the number of vertices between two such path is $O(1)$, we have $O(k^6)$ vertices. Let $\mathcal{P}$ be the set of treepaths that are shrunk to length $O(1)$ in Step 8. Let T' be the outtree obtained from T by shrinking each directed path in $\mathcal{P}$ in the same way. T' is then an outbranching for G' without forward edges. Therefore, edges that are neither tree nor exiting edges must be back edges. (G', T') has $O(k)$ tree edges, $O(|\text{isolated treepaths}|) \cdot 44k^4 + |\text{vertices outside all isolated treepaths}|^2 = O(k^2) \cdot 44k^4 + O(k^2)^2 = O(k^6)$ exiting edges, and $O(k) \cdot O(k^6) = O(k^7)$ edges incident to the $O(k)$ branching vertices. By the third condition of Step 7, there are $< k$ back edges incident to a non-branching vertex. To sum up, G' has $O(k^7)$ edges.

Theorem 13. *In linear time, a kernel with $O(k^6)$ vertices and $O(k^7)$ edges can be found for the* ROOTED k-LEAF OUTBRANCHING PROBLEM.

References

1. Alstrup, S., Harel, D., Lauridsen, P.W., Thorup, M.: Dominators in linear time. SIAM J. Comput. 28, 2117–2132 (1999)
2. van Bevern, R.: Towards optimal and expressive kernelization for d-hitting set. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 121–132. Springer, Heidelberg (2012)
3. van Bevern, R., Hartung, S., Kammer, F., Niedermeier, R., Weller, M.: Linear-time computation of a linear problem kernel for dominating set on planar graphs. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 194–206. Springer, Heidelberg (2012)
4. Binkele-Raible, D., Fernau, H., Fomin, F.V., Lokshtanov, D., Saurabh, S., Villanger, Y.: Kernel(s) for problems with no kernel: On out-trees with many leaves. ACM Transactions on Algorithms 8(4), 38 (2012)
5. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. 25(6), 1305–1317 (1996)
6. Daligault, J., Thomassé, S.: On finding directed trees with many leaves. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 86–97. Springer, Heidelberg (2009)
7. Downey, R.G., Fellows, M.R.: Parameterized complexity. Springer (1999)
8. Hagerup, T.: Simpler linear-time kernelization for planar dominating set. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 181–193. Springer, Heidelberg (2012)
9. Harel, D.: A linear time algorithm for finding dominators in flow graphs and related problems. In: ACM Symp. on Theory of Computing (STOC), vol. 17, pp. 185–194 (1985)
10. Kloks, T., Lee, C.-M., Liu, J.: New algorithms for k-face cover, k-feedback vertex set, and k-disjoint cycles on plane and planar graphs. In: Kučera, L. (ed.) WG 2002. LNCS, vol. 2573, pp. 282–295. Springer, Heidelberg (2002)
11. Protti, F., Dantas da Silva, M., Szwarcfiter, J.: Applying modular decomposition to parameterized cluster editing problems. Theory of Computing Systems 44, 91–104 (2009)