Algorithmica (2016) 76:279–296
DOI 10.1007/s00453-015-0037-3

Parameterized Algorithms for Non-separating Trees and Branchings in Digraphs

Jørgen Bang-Jensen[1] · **Saket Saurabh**[2] · **Sven Simonsen**[1]

Received: 13 August 2014 / Accepted: 14 July 2015 / Published online: 24 July 2015

Abstract A well known result in graph algorithms, due to Edmonds, states that given a digraph D and a positive integer ℓ, we can test whether D contains ℓ arc-disjoint out-branchings in polynomial time. However, if we ask whether there exists an out-branching and an in-branching which are arc-disjoint, then the problem becomes NP-complete. In fact, even deciding whether a digraph D contains an out-branching which is arc-disjoint from some spanning tree in the underlying undirected graph remains NP-complete. In this paper we formulate some natural optimization questions around these problems and initiate its study in the realm of parameterized complexity. More precisely, the problems we study are the following: ARC-DISJOINT BRANCHINGS and NON-DISCONNECTING OUT-BRANCHING. In ARC-DISJOINT BRANCHINGS (NON-DISCONNECTING OUT-BRANCHING), a digraph D and a positive integer k are given as input and the goal is to test whether there exist an out-branching and in-branching (respectively, a spanning tree in the underlying undirected graph) that differ on at least k arcs. We obtain the following results for these problems.

- NON-DISCONNECTING OUT-BRANCHING is fixed parameter tractable (FPT) and admits a linear vertex kernel.
- ARC-DISJOINT BRANCHINGS is FPT on strong digraphs.

✉ Saket Saurabh
saket@imsc.res.in

Jørgen Bang-Jensen
jbj@imada.sdu.dk

Sven Simonsen
svsim@imada.sdu.dk

[1] Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark

[2] The Institute of Mathematical Sciences, Chennai, India

The algorithm for NON-DISCONNECTING OUT-BRANCHING runs in time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ and the approach we use to obtain this algorithms seems useful in designing other moderately exponential time algorithms for edge/arc partitioning problems.

Keywords Branching · Spanning tree · Fixed parameter tractable · Parameterized complexity · Linear vertex kernel · Exponential time algorithm · Partitioning problem

1 Introduction

A digraph T is an *out-tree* (an *in-tree*) if T is an oriented tree with just one vertex s of in-degree zero (out-degree zero). The vertex s is the root of T. If an out-tree (in-tree) T is a spanning subdigraph of D, T is called an *out-branching* (an *in-branching*). We use the notation B_s^+ (B_s^-) to denote an out-branching (in-branching) rooted at s of the digraph. The study of finding a spanning tree in an undirected graph or an out-branching in a digraph satisfying certain properties, such as having at least k leaves, or having at least k internal vertices [1,8–10,13,14,18,19] has been at the forefront of research in parameterized algorithms in the last few years. The goal of this paper is to initiate a study of a different class of problems around spanning trees/out-branchings in the realm of parameterized complexity.

Our starting point is a classical result by Edmonds that states that given a digraph D and a positive integer ℓ, we can test whether D contains ℓ arc-disjoint out-branchings in polynomial time [16]. Given this result, a natural question is whether in polynomial time one can find an out-branching and an in-branching that are arc-disjoint. However, it is NP-complete to decide whether a given digraph has a pair of arc-disjoint branchings B_s^+, B_t^- [2]. In fact, as was shown recently by two of the authors, this already holds for 2-regular digraphs [4]. Similarly, as shown by Edmonds [15], one can also decide in polynomial time whether an undirected graph G contains a pair of edge-disjoint spanning trees and find such trees if they exist. Inspired by these results Thomassé asked around 2005 whether one could decide in polynomial time, for a given input digraph D, whether D contains an out-branching such that deleting the arcs of this would leave the resulting digraph connected in the underlying sense. This was recently answered in the negative by the first author and Yeo [6]. That is, it is NP-complete to decide whether a given digraph $D = (V, A)$ contains an out-branching B_s^+ such that the underlying undirected graph of $D' = (V, A \backslash A(B_s^+))$ is connected. Here, $A(B_s^+)$ denotes the arc set of B_s^+. In this paper we formulate some natural optimization questions around these two problems and initiate their study in the realm of parameterized complexity. In what follows we define the problems we study, the results we get and give an overview of the related results.

For our first problem we need to define spanning trees on digraphs. A **spanning tree** of a digraph D is a spanning subdigraph T such that the underlying undirected graph of T is a spanning tree of the underlying undirected graph of D.

Non-Disconnecting Out-Branching. The first problem we study is as follows.

NON-DISCONNECTING OUT-BRANCHING (NDOB) **Parameter:** k
Input: A digraph $D = (V, A)$ and a non-negative integer k.
Question: Does D have an out-branching B_s^+ and a spanning tree T of D such that $|A(B_s^+)\backslash A(T)| \geq k$?

This problem is a parameterized version of the problem of deciding whether D contains an out-branching such that deleting the arcs of this would leave the resulting digraph connected in the underlying sense. Another parameterization for this problem could be whether there is an out-branching B_s^+ and a spanning tree T of D so that they have at most k arcs in common. However, observe that this problem becomes NP-complete even for $k = 0$ and thus is not FPT.

For NDOB we first obtain a linear vertex kernel and then design an FPT algorithm for this problem running in time $2^{\mathcal{O}(k)}|V|^{\mathcal{O}(1)}$. It is important to note that our FPT algorithm does not follow directly from our linear vertex kernel and uses a randomized separation argument.

Arc-Disjoint Branchings. The second problem we study is as follows.

ARC-DISJOINT BRANCHINGS **Parameter:** k
Input: A digraph $D = (V, A)$ and a non-negative integer k.
Question: Does D have branchings B_s^+ and B_t^- such that $|A(B_s^+)\backslash A(B_t^-)| \geq k$?

It is a parameterized version of the problem of deciding whether a given digraph has a pair of arc-disjoint branchings B_s^+, B_t^-. We show that ARC-DISJOINT BRANCHINGS is FPT on strong digraphs and strongly believe that our algorithm can be extended to all digraphs. To derive our result, we show that in polynomial time either we can correctly decide whether the input is a YES-instance, or obtain a tree-decomposition of width at most $3k$ for the underlying undirected graph. Now we get the desired FPT algorithm by standard dynamic programming over graphs of bounded treewidth.

Bang-Jensen and Yeo [5] studied the related MINIMUM SPANNING STRONG SUBDIGRAPH (MSSS) problem. MSSS is the problem of finding, in a strong digraph $D = (V, A)$, a spanning strong subdigraph $D' = (V, A')$ where $A' \subseteq A$ and $|A'|$ is minimum. This problem is NP-hard as it contains the hamiltonian cycle problem as a special case. Every strong digraph contains an out-branching B_s^+ and an in-branching B_s^- and the union of B_s^+ and B_s^- is a strong spanning subdigraph of D, hence the optimum A' has size at most $2|V| - 2$. It was shown in [5] that MSSS is FPT when we use the distance from the upper bound $2|V| - 2$ as a parameter. That is, there exists an $f(k)|V|^{\mathcal{O}(1)}$ algorithm for deciding whether a given strong digraph D on n vertices contains a spanning strong subdigraph with at most $2n - 2 - k$ arcs. The parameterized version of MSSS is equivalent to deciding for a given strong digraph D and parameter k whether there exists an out-branching B_s^+ and an in-branching B_s^- such that $|A(B_s^+) \cap A(B_s^-)| \geq k$. If, instead of maximizing the intersection between the branchings, we seek to minimize the intersection, we get ARC-DISJOINT BRANCHINGS. In [5] it was asked whether ARC-DISJOINT BRANCHINGS is FPT. We answer this question in the affirmative for strong digraphs.

Vertex Exponential Exact Algorithms for Arc-Disjoint Problems. The approach used in designing a $2^{\mathcal{O}(k)}|V|^{\mathcal{O}(1)}$ time algorithm for NDOB seems of general interest and we outline a scheme for producing vertex exponential exact algorithms for a large class of related problems. For discussion let us focus on the problem of deciding whether a given digraph $D = (V, A)$ has a pair of arc-disjoint branchings B_s^+, B_t^-. A trivial enumerative algorithm will guess arc sets for B_s^+ and B_t^- and then check whether they are disjoint. This algorithm will run in time $m^{\mathcal{O}(n)}$, where $m = |A|$ and $n = |V|$. However, we obtain an algorithm running in time $2^{\mathcal{O}(n)}$ for this and many similar problems. It is important to point out that an exact algorithm for MSSS running in time $2^{\mathcal{O}(n)}$ was only recently obtained using tools from matroids [20].

2 Preliminaries

An instance of a parameterized problem consists of (x, k), where k is called the parameter. A central notion in parameterized complexity is *fixed parameter tractability* (FPT) which means, for a given instance (x, k), solvability in time $f(k) \cdot p(|x|)$, where f is an arbitrary function of k and p is a polynomial in the input size. The notion of *kernelization* is defined as follows.

Definition 1 (*Kernelization*) Let Π be a parameterized problem and g be a computable function. We say that Π *admits a kernel of size* g if there exists an algorithm $\mathbb{K}$, called a *kernelization algorithm*, or, in short, a *kernelization*, that given $(x, k) \in \Pi$, outputs, in time polynomial in $|x| + k$, a pair (x', k') such that

(a) $(x, k) \in \Pi$ if and only if $(x', k') \in \Pi$, and
(b) $\max\{|x'|, k'\} \leq g(k)$.

When $g(k) = k^{O(1)}$ or $g(k) = O(k)$ then we say that Π *admits a polynomial* or *linear kernel* respectively.

We say that a graph problem admits a linear vertex kernel if the number of vertices in the reduced graph is bounded by a linear function of k.

Let $D = (V, A)$ be a digraph. By $V(D)$ and $A(D)$ we represent the vertex set and the arc set of D, respectively. Given a subset $V' \subseteq V(D)$ of a digraph D, let $D[V']$ denote the digraph induced by V'. The *underlying graph* $\mathrm{U}(D)$ of D is obtained from D by omitting all orientations of arcs. A digraph D is *strong* if, for every pair x, y of vertices there are directed paths from x to y and from y to x. A vertex u of D is an *in-neighbor* (*out-neighbor*) of a vertex v if $uv \in A(D)$ ($vu \in A(D)$, respectively). The *in-degree* $d^-(v)$ (*out-degree* $d^+(v)$) of a vertex v is the number of its in-neighbors (out-neighbors). For an arc uv, we call u the arc's *tail* and v its *head*.

Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a pair $(T, \mathcal{X} = \{X_t\}_{t \in V(T)})$ where T is a tree and $\mathcal{X}$ is a collection of subsets of V such that:

- $\forall e = uv \in E,\ \exists t \in V(T) : \{u, v\} \subseteq X_t$ and
- $\forall v \in V,\ T[\{t \mid v \in X_t\}]$ is non-empty and connected.

We call the vertices of T *nodes* and the sets in $\mathcal{X}$ *bags* of the tree decomposition $(T, \mathcal{X})$. The *width* of $(T, \mathcal{X})$ is equal to $\max\{|X_t| - 1 \mid t \in V(T)\}$ and the *treewidth* of $G = (V, E)$ is the minimum width over all tree decompositions of G.

Throughout the paper we use n and m to denote the number of vertices and arcs in the input digraph. Notation not given here is consistent with [3].

3 Non-Disconnecting Out-Branching

In this section we first give a vertex kernel with at most $12k$ vertices for a rooted version of the problem, then design a vertex kernel for NDOB with at most $20k$ vertices. Finally, using the kernel for the rooted version we design a $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time algorithm for NDOB.

3.1 A Kernel for Rooted NDOB

In this section we first define a rooted version of the problem and obtain a vertex kernel wit at most $12k$ vertices which allows us to design a faster FPT algorithm for NDOB. The problem is defined as follows.

Rooted NDOB	**Parameter:** k
Input: A digraph D, a root $s \in V(D)$ and a non-negative integer k.	
Question: Does D have a branching B_s^+ and a spanning tree T in D such that $\lvert A(B_s^+) \backslash A(T)\rvert \geq k$?	

To obtain our vertex kernel for Rooted NDOB with at most $12k$ vertices we first introduce some simple reduction rules which we use to bound the size of the kernel.

Rule 1 If D has no out-branching rooted at s replace D by an independent set of size 2.

Rule 2 If D has a vertex $v \in V$ with $d^-(v) = 1$ and $d^+(v) = 0$ and $v \neq s$, delete v.

We say we **contract** an arc (edge) uv when we delete uv and identify u with v keeping all their adjacent arcs (edges).

Rule 3 If D has an arc $uv \in A$ such that $d^-(u) = d^+(u) = d^-(v) = 1$ and $u \neq s \neq v$, contract uv.

A reduction rule is **safe** with respect to a problem if applying the rule does not change the answer to the problem, that is after applying a safe rule with respect to problem A the answer to problem A is YES if and only if the answer was YES before the rule was applied.

Rule 1 is clearly safe. We show below that the other two are also safe.

Lemma 2 *Rules 2 and 3 are safe and can be applied in polynomial time.*

Proof All structures in rules 2 and 3 can be identified in linear time and performing an update takes constant time. So we are left with arguing that all rules are safe. Rule 2 is safe since every spanning tree and out-branching must contain the unique arc incident to the vertex v.

Now we show that Rule 3 is safe. Let (D, k) be the given instance and (D', k) be the instance obtained after applying contraction to the arc uv. Let the contracted vertex be z_{uv}. We show that (D, k) is a YES-instance if and only if (D', k) is a YES-instance. We first show the forward direction of the proof. Let B_s^+ be an out-branching rooted at s and T be a spanning tree of D such that $|A(B_s^+)\backslash A(T)| \geq k$. Let w be the in-neighbor of u. We know that both arcs wu and uv appear in B_s^+. Let B_s^{+c} be the out-branching of D' obtained after contracting the arc uv into a vertex z_{uv} in B_s^+. If uv also appears in T then we just contract the edge uv and obtain a spanning tree T' of D'. Since uv is present in both B_s^+ and T we have that $|A(B_s^{+c})\backslash A(T')| \geq k$. Now we assume that uv does not appear in T. Observe that u has degree 2 in $U(D)$. Since the edge uv is not present in T we have that the edge wu is present in T. Now obtain a spanning tree T' in D by deleting the edge wu and adding the edge uv to T. Clearly, if $|A(B_s^+)\backslash A(T)| \geq k$ then we have that $|A(B_s^+)\backslash A(T')| \geq k$ and this case reduces to the previous case.

Now we show the reverse direction of the proof. Suppose we have B_s^{+c} and T' in D' such that $|A(B_s^{+c})\backslash A(T')| \geq k$. If w is a neighbor to z_{uv} in T' then we obtain T by modifying T' as follows: delete z_{uv}, introduce u and v, make w adjacent to u, add the edge uv, and make every other vertex than w that was adjacent to z_{uv} now adjacent to v. If w is not a neighbor to z_{uv} in T' then we obtain T from T' by renaming z_{uv} to v and adding the edge uv to it. We obtain B_s^+ from B_s^{+c} by deleting the vertex z_{uv}, adding u and v, adding the arc wu (we have the arc wz_{uv} in B_s^{+c} since $d^-_{D'}(z_{uv}) = 1$) and making v in-neighbor to every other vertex z_{uv} was in-neighbor to. Since we have not deleted any arc/edge from B_s^{+c} and T' we have that $|A(B_s^+)\backslash A(T)| \geq k$. This concludes the proof. □

We call an instance (D, k) **reduced** according to a reduction rule if the rule can not be applied to (D, k) anymore. We prove that reduction according to these rules and a greedy construction algorithm are sufficient to give a linear vertex kernel.

Lemma 3 *Let (D, k) be an instance to* ROOTED NDOB. *Then in polynomial time either we*

- *conclude that (D, k) is a* NO*-instance;*
- *or find an out-branching B_s^+ and a spanning tree T such that $|A(B_s^+)\backslash A(T)| \geq k$;*
- *or find a vertex kernel with at most $12k$ vertices.*

Proof If Rule 1 applies we answer NO. If not we reduce (D, k) according to Rules 2 and 3. After this we find an arbitrary out-branching B_s^+ rooted at s. Such a branching must exist since otherwise Rule 1 would apply and B_s^+ can be found in $O(m)$ time using one BFS-search.

Greedy-Tree-Update
Let T be the spanning tree with the same arc-set as B_s^+. Furthermore, let $W = A(D)\backslash A(B_s^+)$, $\mathsf{diff}(T) = \emptyset$ and $\mathsf{diff}(B_s^+) = \emptyset$. Repeat the following until no longer possible.

If there exists an arc $a \in W$ such that the fundamental cycle $C(T, a)$ of $T + a$ shares an arc with B_s^+ (that is there exists an arc $a' \in A(C(T, a)) \cap A(B_s^+)$) then update the spanning tree and other sets as follows.

- $T := T + a - a'$;
- $W := W \backslash \{a\}$;
- $\mathsf{diff}(T) := \mathsf{diff}(T) \cup \{a\}$ and $\mathsf{diff}(B_s^+) := \mathsf{diff}(B_s^+) \cup \{a'\}$.

Return T.

Let B_s^+ still be our arbitrarily chosen out-branching and T be the spanning tree returned by **Greedy-Tree-Update**. If $|\mathsf{diff}(B_s^+)| \geq k$ we have our solution and we return B_s^+ and T. So let us assume that $|\mathsf{diff}(B_s^+)| < k$ which implies $|\mathsf{diff}(T)| < k$.

Let X be the set of endvertices of the arcs in $\mathsf{diff}(B_s^+) \cup \mathsf{diff}(T)$. The size of X is at most $2(k-1) + 2(k-1) = 4k - 4$ and every arc in $A \backslash A(B_s^+)$ has both endvertices in X from B_s^+: otherwise, if there was an arc a with at least one end in $V \backslash X$ we could have continued one more step above, as B_s^+ and T are identical in this part of the digraph so $C(T, a)$ would have to share arcs with B_s^+.

If we delete all arcs that have both endvertices in X from D, we obtain a collection of disjoint out-trees $F_{s_1}^+, \ldots, F_{s_p}^+$, where $s_1 = s$ and $s_i \in X$ for $2 \leq i \leq p$. This holds because the remaining arcs are all in $A(T) \cap A(B_s^+)$ so we are essentially breaking apart B_s^+.

Partition the vertex set of every $F_{s_i}^+$ into 3 sets: $X_{s_i} = X \cap V(F_{s_i}^+)$, $2_{s_i} = \{v \in (V(F_{s_i}^+) \backslash X_{s_i}) | d^+_{F_{s_i}^+}(v) \geq 2\}$ and $1_{s_i} = \{v \in (V(F_{s_i}^+) \backslash X_{s_i}) | d^+_{F_{s_i}^+}(v) = 1\}$. Observe that $F_{s_i}^+$ can have no leaves in $V \backslash X$ since in that case we could apply Rule 2. So the leaves of $F_{s_i}^+$ are all in X_{s_i} thus upper bounding the number of leaves by $|X_{s_i}|$ and implying that $|2_{s_i}| < |X_{s_i}|$. Likewise $F_{s_i}^+$ can contain no arc uv such that $d^-_{F_{s_i}^+}(u) = d^+_{F_{s_i}^+}(u) = d^-_{F_{s_i}^+}(v) = 1$ and $u, v \notin X$ since then the arcs adjacent to u and v would all be in B_s^+ and we could apply Rule 3. So every vertex in 1_{s_i} must have its only out-neighbor in X_{s_i} implying that $|1_{s_i}| \leq |X_{s_i}|$. Furthermore since the $F_{s_i}^+$ were disjoint we know that the X_{s_i} are disjoint. Collecting all of this we get that

$$|V| = \sum_{i=1}^{p} |V(F_{s_i}^+)| = \sum_{i=1}^{p} \left(|X_{s_i}| + |2_{s_i}| + |1_{s_i}| \right) < \sum_{i=1}^{p} 3|X_{s_i}| \leq 3|X| \leq 12k - 12.$$

So D is a vertex kernel with at most $12\,k$ vertices and we conclude our proof. □

The kernel for NDOB is obtained using reduction rules similar to Rules 1–3.

3.2 Kernel for Non-Disconnecting Out-Branching

In this section we present a polynomial algorithm for obtaining a vertex kernel for NDOB with at most $20\,k$ vertices.

First we introduce some simple reduction rules which we will use to bound the number of vertices of the kernel.

Rule 4 If a vertex $v \in V$ has in-degree 0, fix v as the root and apply Lemma 3.

Rule 5 If D has a vertex $v \in V$ with $d^-(v) = 1$ and $d^+(v) = 0$, delete v.

Rule 6 If D has no out-branching, answer NO.

Rules 4 and 6 are safe and can be checked in polynomial time. Rule 5 is the same as Rule 2 from our section on ROOTED NDOB. This rule is still safe as a vertex with only in-neighbors can never be the root of an out-branching anyways and the unique arc incident with v must be both in all out-branchings and all spanning trees. However, Rule 3 does not apply to NDOB in its current form, as evidenced by the case where u or v are possible roots. To fill its role we introduce the following "longer path" rule.

Rule 7 If D has a path $xuvw$ such that $d^-(u) = d^+(u) = d^-(v) = d^+(v) = d^-(w) = 1$, contract uv.

The extra arc this rule requires compared to Rule 3 allows us to show.

Lemma 4 *Rule 7 is safe and can be applied in polynomial time.*

Proof The 3-path can be identified in polynomial time and performing an update takes constant time. So we are left with arguing that the rule is safe.

Let (D, k) be the given instance and (D', k) be the instance obtained after contracting the arc uv into the vertex z_{uv}. We show that (D, k) is a YES-instance if and only (D', k) is a YES-instance.

We first show the backward direction of the proof. Let B^{+c} be an out-branching and T' be a spanning tree in D' such that $|A(B^{+c})\backslash A(T')| \geq k$. We obtain a solution B^+ and T for D from B^{+c} and T' by doing the following to both. Split z_{uv} into u and v such that u keeps the possible in-arc of z_{uv} while v keeps the possible out-arc, then add the arc uv. Since we have not deleted any arc/edge from B_s^{+c} and T' we have that $|A(B_s^+)\backslash A(T)| \geq k$.

Forward direction. Let B^+ be an out-branching and T be a spanning tree in D such that $|A(B^+)\backslash A(T)| \geq k$, we can construct a solution for D' as follows. First we observe that if $uv \in B^+$ and $uv \in T$ we can just contract it in both and get an out-branching B^{+c} and a spanning tree T' in D' such that $|A(B^{+c})\backslash A(T')| \geq k$. If $uv \notin B^+$ or $vw \notin B^+$ this implies that v or w is the root of B^+. Change the root to u by removing xu from B^+ and adding uv and vw. This change might decrease $|A(B^+)\backslash A(T)|$ by one if $xu \notin T$, but we fix this in the next step. If $xu \notin T$ or $uv \notin T$ add this arc to T then remove vw to break the fundamental cycle. This change increases $|A(B^+)\backslash A(T)|$ by one again if it was decreased in the previous step. Now B^+, T is again a solution in D and uv is contained in both, so we can contract uv and get a solution for D'. □

With this new reduction rule we are able to prove a linear kernel for NON-DISCONNECTING OUT-BRANCHING as well.

Theorem 5 *Let (D, k) be an instance to* NDOB. *Then in polynomial time either we*

- *conclude that (D, k) is a* NO*-instance;*
- *or find an out-branching B^+ and a spanning tree T such that $|A(B^+)\backslash A(T)| \geq k$;*
- *or find a kernel with at most $20k$ vertices.*

Proof If Rule 4 applies we hand the instance with its new root over to the algorithm of Lemma 3 and answer what the algorithm would answer. If Rule 6 applies we answer

NO. If both of these didn't apply we reduce (D, k) according to Rules 5 and 7. After this we find an arbitrary out-branching B_s^+ rooted at some vertex $s \in V$. Such a branching must exist since otherwise Rule 6 would apply and B_s^+ can be found in $O(nm)$ time using one BFS-search for every possible root.

Let T be the spanning tree returned by the procedure **Greedy-Tree-Update** from Lemma 3 run on our arbitrarily chosen out-branching B_s^+. As before we can assume that $|\mathsf{diff}(B_s^+)| = |\mathsf{diff}(T)| < k$. Again we let X be the set of endvertices of the arcs in $\mathsf{diff}(B_s^+) \cup \mathsf{diff}(T)$ and observe that $|X| \leq 2(k-1) + 2(k-1) = 4k-4$ and since the update rule of **Greedy-Tree-Update** could no longer be applied every arc in $A \backslash A(B_s^+)$ has both endvertices in X from B_s^+.

If we delete all arcs that have both endvertices in X from D, we obtain a collection of disjoint out-trees $F_{s_1}^+, \ldots, F_{s_p}^+$, where $s_1 = s$ and $s_i \in X$ for $2 \leq i \leq p$.

Partition the vertex set of every F_{s_i} into 3 sets: $X_{s_i} = X \cap V(F_{s_i})$, $2_{s_i} = \{v \in (V(F_{s_i}) \backslash X_{s_i}) | d^+_{F_{s_i}}(v) \geq 2\}$ and $1_{s_i} = \{v \in (V(F_{s_i}) \backslash X_{s_i}) | d^+_{F_{s_i}}(v) = 1\}$. Then further partition 1_{s_i} into two parts, let $1^X_{s_i}$ be the set of vertices of 1_{s_i} that have their out-neighbor in X_{s_i} and let $1^{V \backslash X}_{s_i} = 1_{s_i} \backslash 1^X_{s_i}$. Since D was reduced according to Rule 5 we know that all leaves of F_{s_i} are in X_{s_i} so $|2_{s_i}| < |X_{s_i}|$. By construction every vertex in $1^X_{s_i}$ must have its only out-neighbor in X_{s_i} so $|1^X_{s_i}| \leq |X_{s_i}|$, since no two vertices in F_{s_i} can have the same out-neighbor. Now to bound the size of $1^{V \backslash X}_{s_i}$ observe that the out-neighbor of any vertex in $1^{V \backslash X}_{s_i}$ can't be in $1^{V \backslash X}_{s_i}$ as well, since then Rule 7 would apply, so the out-neighbor must be either in 2_{s_i} or in $1^X_{s_i}$. This gives $|1^{V \backslash X}_{s_i}| \leq |2_{s_i}| + |1^X_{s_i}| \leq 2|X_{s_i}|$. Collecting all of this we get that

$$
\begin{aligned}
|V| &= \sum_{i=1}^{p} |V(F_{s_i})| = \sum_{i=1}^{p} \left(|X_{s_i}| + |2_{s_i}| + |1^X_{s_i}| + |1^{V \backslash X}_{s_i}| \right) \\
&< \sum_{i=1}^{p} 5|X_{s_i}| \leq 5|X| \leq 20k - 20.
\end{aligned}
$$

So D is a kernel for NDOB with at most $20\,k$ vertices and we conclude our proof. □

3.3 Faster FPT Algorithm for Non-Disconnecting Out-Branching

Here, we give a $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ time algorithm for NDOB. For our main algorithm we need the following subroutine.

Lemma 6 *Given an undirected graph G, a coloring function $f : E(G) \to \{0, 1\}$, and a non-negative integer k, we can find in linear time a spanning tree T such that it contains at least k edges colored 0 or decide that no such tree exists.*

Proof First of all we assume that the graph G is connected, else we know that G does not have any spanning tree. Let E_0 be the set of edges that have been colored 0. Consider the graph G' with the vertex set $V(G)$ and the edge set E_0. Find a spanning forest F' of G' with largest possible number of edges. Now since G is connected F'

can be extended to a spanning tree T of G. Thus, G has a spanning tree with at least k edges colored 0 if and only if G' has a spanning forest on at least k edges. Since our algorithm just runs two rounds of any standard algorithm for finding a maximum forest (which can be done with simple Depth-First-Search) we have that the algorithm runs in linear time. □

Now we design a randomized algorithm for NDOB.

Theorem 7 *There is a one-sided-error Monte-Carlo algorithm for* NDOB *running in* $\mathcal{O}((33.97)^k \cdot (n+m))$ *expected time.*

Proof Given an input (D, k) to NDOB, the algorithm goes through all n vertices as potential root and applies the following procedure. If for any vertex currently chosen as a root candidate we succeed in finding the desired solution we return YES else we return NO.

Given a root first apply the kernelization algorithm described in Lemma 3. The algorithm, in polynomial time, either returns a solution to ROOTED NDOB and thus to NDOB, in which case we answer YES, or it returns an equivalent instance (D', k) such that $|V(D')| \leq 12k$. In the second case we perform the following steps several times.

1. Uniformly at random color the arcs of D' with $[13k] = \{1, 2, \ldots, 13k\}$. Let the coloring function be called f.
2. Using Lemma 6 test whether the underlying undirected graph of D' contains a spanning tree with at least k edges colored with colors from the set $\{1, \ldots, k\}$ by treating all these colors as 0 and others as 1. Also test whether the digraph D'' on the vertex set $V(D')$ and the arc set containing all those arcs that have been colored $\{k+1, \ldots, 13k\}$ contains an out-branching. If the answer to both these questions is YES then return YES.

Given a spanning tree T and an out-branching B_s^+, we define $\mathsf{diff}(T) = A(T) \setminus A(B_s^+)$ and $\mathsf{diff}(B_s^+) = A(B_s^+) \setminus A(T)$. The answer to our problem is YES if and only if there exists a tree T and out-branching B_s^+ such that $|\mathsf{diff}(T)| \geq k$ and thus $|\mathsf{diff}(B_s^+)| \geq k$. We say that the solution (B_s^+, T) is **colorful** with respect to f, if

- every arc of B_s^+ has been colored with $\{k+1, \ldots, 13k\}$ and
- some subset $S \subseteq \mathsf{diff}(T)$ of size k has been colored with $\{1, \ldots, k\}$.

The probability that a solution (B_s^+, T) becomes colorful is:

$$\sum_{\{S \subseteq \mathsf{diff}(T), |S|=k\}} (12k/13k))^{|V(D')|-1} (k/13k)^k \geq ((12/13)^{12} \cdot (1/13))^k.$$

Given a coloring, Step 2 of the algorithm finds a colorful solution if one exists, in polynomial time. The correctness of the algorithm follows from the fact that both T and B_s^+ have $|V(D')|-1$ arcs and if T contains at least k arcs that are of different color than the arcs of B_s^+ then we have that $|\mathsf{diff}(T)| \geq k$ and consequently $|\mathsf{diff}(B_s^+)| \geq k$ since T and B_s^+ have the same number of arcs. Thus, if there is indeed a solution (B_s^+, T) then we find one with probability at least $\left(\left(\frac{12}{13}\right)^{12} \cdot \frac{1}{13}\right)^k$. Thus, repeating the

algorithm $\left(\left(\frac{13}{12}\right)^{12} \cdot 13\right)^k \leq 33.97^k$ times, the error probability can be reduced to at most $1/e$ meaning that the success probability is at least $1 - \frac{1}{e} \geq \frac{1}{2}$. □

We can obtain a deterministic algorithm for the problem using the notion of *lopsided universal family*.

Definition 8 ([20,22]). An *n-p-q-lopsided-universal* family $\mathcal{F}$ is a set of functions from $\{1, \ldots, n\}$ to $\{0, 1\}$, such that for every subset $A, B \subseteq \{1, \ldots, n\}$, $A \cap B = \emptyset$, $|A| = p$ and $|B| = q$, there exists a function $f \in \mathcal{F}$ such that for every $a \in A$, $f(a) = 0$ and for every $b \in B$, $f(b) = 1$.

Theorem 9 ([20,22]). *There is an algorithm that given n, p and q constructs a n-p-q-lopsided-universal family $\mathcal{F}$ of size $\binom{p+q}{p} \cdot 2^{o(p+q)} \cdot \log n$ in time $\mathcal{O}(\binom{p+q}{p} \cdot 2^{o(p+q)} \cdot n \log n)$.*

Theorem 10 *There is an algorithm for* Non-Disconnecting Out-Branching *running in $\mathcal{O}(33.97^{k+o(1)} \cdot (n+m) \log n)$ time.*

Proof To obtain the deterministic algorithm we change the algorithm suggested in Theorem 7 so that it takes coloring functions from a universal family instead of creating them randomly. To achieve this we replace the two repeated steps in the algorithm with the following. Let m' and n' denote the number of arcs and the number of vertices in the reduced instance (D', k).

1. Label the arcs of D' $1, 2, \ldots, m'$ then for every $f \in \mathcal{F}$, where $\mathcal{F}$ is an $(m', k, n' - 1)$-lopsided-universal family, do as follows:
   - Using Lemma 6 test whether the underlying undirected graph of D' contains a spanning tree with at least k edges colored with 0 under f. Also test whether the digraph D'' on the vertex set $V(D)$ and the arc set containing all those arcs that has been colored 1 contains an out-branching rooted at s. If the answer to both these questions is Yes then return Yes.
2. Return that the original instance is a No-instance.

The correctness of the algorithm follows from the following. Let (B^+, T) be a solution and W be $A(B^+) \cap S$, where S is some fixed set of k arcs in $\mathsf{diff}(T)$. Since the size of $A(B^+)$ is $n' - 1$ there exists a function $f \in \mathcal{F}$ such that f colors all the arcs of B^+ with 1 and the arcs of S with 0. The running time of the algorithm is dominated by the size of the set of hash functions, which by Theorem 9 is upper bounded by $\binom{13k}{k} 2^{o(k)}$. To upper bound $\binom{13k}{k}$, we will use the following well known identity on binomial coefficients:

$$\binom{n}{k} \leq \frac{n^n}{k^k (n-k)^{n-k}}.$$

A direct application of the above identity implies

$$\binom{13k}{k} \leq \left(\left(\frac{13}{12}\right)^{12} \cdot 13\right)^k \leq 33.97^k.$$

This concludes the proof. □

4 ARC-DISJOINT BRANCHINGS on Strong Digraphs

In this section we consider ARC-DISJOINT BRANCHINGS on strong digraphs and design an FPT algorithm for this. We do this by showing that in polynomial time either we can find an out-branching and an in-branching that differ on at least k arcs or we can find a tree-decomposition of the underlying undirected graph of width at most $3k$. The main technical lemma of this section is as follows.

Lemma 11 *Let (D, k) be an instance to* ARC-DISJOINT BRANCHINGS *where D is a strong digraph. Then in polynomial time either we*

- *conclude that (D, k) is a* NO*-instance;*
- *or find an out-branching B_s^+ and an in-branching B_t^- such that $|A(B_s^+) \setminus A(B_t^-)| \geq k$;*
- *or find a tree-decomposition of width $3k - 2$ for $U(D)$.*

Proof We start with an arbitrary pair of an out-branching B_s^+ and an in-branching B_t^-, rooted at s and t respectively where s and t could potentially be the same vertex. Such branchings can be constructed by two BFS-searches in polynomial time and since D is strong they always exist. So from now onwards we assume that we have an out-branching B_s^+ and an in-branching B_t^-. Next we describe a procedure that finds a pair of branchings with nice structural properties.

Procedure-Local-Optimal
Let B_s^+ and B_t^- be the branchings found in the first paragraph. Furthermore, let $W = A(D) \setminus (A(B_s^+) \cup A(B_t^-))$, $\mathsf{diff}(B_s^+) = A(B_s^+) \setminus A(B_t^-)$ and $\mathsf{diff}(B_t^-) = A(B_t^-) \setminus A(B_s^+)$ at all times throughout the procedure. Repeat the following until no longer possible.

If there exists an arc $a = uv \in W$ such that $B_s^+ + a$ (that is add the arc a to the out-branching B_s^+) contains no directed cycle and the unique in-arc a' of v in B_s^+ satisfies $a' \notin \mathsf{diff}(B_s^+)$ then update the out-branching and other sets as follows.

- $B_s^+ := B_s^+ + a - a'$;
- $W := W \setminus \{a\}$;
- $\mathsf{diff}(B_s^+) := \mathsf{diff}(B_s^+) \cup \{a\}$ and $\mathsf{diff}(B_t^-) := \mathsf{diff}(B_t^-) \cup \{a'\}$.

Return B_s^+ and B_t^-.

Let B_s^+ and B_t^- be the out-branching and in-branching returned by **Procedure-Local-Optimal**. If $|\mathsf{diff}(B_s^+)| \geq k$ we have our solution and we return B_s^+ and B_t^-. So let us assume that $|\mathsf{diff}(B_s^+)| < k$. This implies that $|\mathsf{diff}(B_t^-)| < k$. We use these trees to find a tree-decomposition of width $3k$ for $U(D)$.

We say an arc uv goes **backwards** with respect to a branching B, if B contains a (v, u)-path, that is, adding the arc to the branching would create a directed cycle. Let $H_{\mathsf{diff}(B_s^+)}$ be the set of vertices that are head of the arcs in $\mathsf{diff}(B_s^+)$. Clearly, $|H_{\mathsf{diff}(B_s^+)}| \leq k - 1$. Observe that every arc of W is either a backward arc with respect to B_s^+ or has its head in $H_{\mathsf{diff}(B_s^+)}$. Let W_h be the set of arcs in W whose heads are in $H_{\mathsf{diff}(B_s^+)}$.

Let $X = \mathsf{diff}(B_t^-) \cup W_h$ and H_X be the set of vertices that are head of the arcs in X. Since, $|\mathsf{diff}(B_t^-)| < k$ we have that $|H_X| \leq 2(k-1)$. Let D' be the digraph obtained by deleting the arcs in X. Observe that every arc of B_s^+ is still in D' and thus it is also an out-branching rooted at s in D'. Let T_v^+ denote the sub-out-branching of B_s^+ rooted at v and let H_v denote the heads of arcs in $A(D')$ that have tail in $V(T_v^+)$ and head in $V(D')\backslash V(T_v^+)$. If for any v we have that $|H_v| \geq k$, then we construct an out-branching B_v^+ in D such that $|A(B_v^+) - A(B_t^-)| \geq k$ as follows.

> Start with the sub-out-branching T_v^+ of B_s^+ rooted at v. Now for every vertex $w \in H_v$ pick an arbitrary arc a_w with tail in $V(T_v^+)$ and head being w. Let this set of arcs be Y. Now add Y to T_v^+. That is, let $T_v^+ := T_v^+ \cup Y$.

Clearly, T_v^+ constructed above is an out-tree and furthermore since $Y \subseteq W\backslash W_h$ we have that $|A(T_v^+)\backslash A(B_t^-)| \geq k$. Now since D is a strong digraph we can extend T_v^+ to an out-branching B_v^+ in D. Thus, in this case we have our solution and we return B_v^+ and B_t^-. So from now onwards we assume that for every $v \in V(D')$, $|H_v| \leq k-1$.

Claim 1 The treewidth of $U(D')$ is at most k.

Proof Let L denote the underlying undirected tree rooted at s corresponding to B_s^+. Furthermore, for $v \neq s$, let $p(v)$ denote the parent of v in B_s^+. That is, $p(v)$ is the unique in-neighbor of v in B_s^+. Let L_v denote the subtree of L rooted at v. Observe that arcs of D' are either arcs of B_s^+ or backward arcs corresponding to B_s^+. Thus, we have that $N_{U(D')}(V(L_v)) = H_v \cup \{p(v)\}$. This in turn implies that $|N_{U(D')}(V(L_v))| \leq k$, so no bag has more than $k+1$ vertices.

We construct a tree decomposition $(T, \mathcal{X} = \{X_t\}_{t \in V(T)})$ of $U(D')$ as follows. We take $T := L$, $X_s = \{s\}$ and for every $v \neq s$, take $X_v = \{v\} \cup N_{U(D')}(V(L_v))$.[1] Every vertex and edge of $U(D')$ belong to some bag, since D' only contains B_s^+ and backwards arcs. Now we need to show that for each vertex $v \in V(U(D'))$ the subgraph $T[\{t \mid v \in X_t\}]$ is connected. Observe that v can only belong to X_w where $w \in V(L_v)$ and the only reason it belongs to X_w for some $w \neq v$ is that $v \in H_w$ or $v = p(w)$. However, once it belongs to H_w for some w then it belongs to every w' on the path from w to v. This proves that $T[\{t \mid v \in X_t\}]$ is connected. Thus $(T, \mathcal{X})$ is a tree decomposition of width at most k, implying by definition that the treewidth of $U(D')$ is at most k. This concludes the proof of the claim. □

Finally, we show how to obtain a tree decomposition for $U(D)$. Recall that, $X = \mathsf{diff}(B_t^-) \cup W_h$ and H_X is the set of vertices that are head of the arcs in X and D' is the digraph obtained by deleting the arcs in X. We also have that $|H_X| \leq 2(k-1)$. We obtain the tree decomposition of $U(D)$ by taking the tree decomposition $(T, \mathcal{X} = \{X_t\}_{t \in V(T)})$ of $U(D')$ and adding H_X to every bag. That is, $X_t := X_t \cup H_X$ for all v in $V(T)$, this way we ensure that every edge of X is in some bag without violating the already established tree decomposition. The resulting tree decomposition has width at most $k + |H_X| \leq 3k-2$. This completes the proof. □

Now we get the desired FPT algorithm by doing the standard dynamic programming over graphs of bounded treewidth. However, this would take $2^{\mathcal{O}(t \log t)} n^{\mathcal{O}(1)}$ and thus

[1] Where $N_{U(D')}(V(L_v))$ is the neighborhood of $V(L_v)$ in $U(D')$.

getting an algorithm for Arc-Disjoint Branchings on strong digraphs running in time $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$. Alternatively, the property of containing an out-branching and an in-branching that differs on at least k arcs can be formulated as a monadic second order formula. Thus, by the fundamental theorem of Courcelle [11,12], the problem Arc-Disjoint Branchings for all digraphs D with treewidth at most $\mathcal{O}(k)$ can be solved in $f(k) \cdot n$ time, where f is a function depending only on k. Thus, we have the following theorem.

Theorem 12 Arc-Disjoint Branchings *is FPT on strong digraphs.*

5 Exact Algorithms for Some Subdigraph Problems

In this section we abstract the ideas used in the algorithm for NDOB deriving a method that is applicable to a range of edge partitioning problems satisfying some constraints. We then apply this method to a number of related problems.

The idea of randomly partitioning arcs to solve NDOB seems to be a promising approach for other edge/arc partitioning problems in the realm of moderately exponential time algorithms. We refer to the book of Fomin and Kratsch [17] for a detailed introduction to the subject.

The class of digraphs on wich our method can be applied is defined as follows:

Definition 13 A class of (di)graphs Π is called **good** if the following holds.

(a) There exist linear functions b_π so that every (di)graph $H \in \Pi$ contains a spanning sub(di)graph H' such that $H' \in \Pi$ and $|A(H')| \leq b_\pi(|V(H)|)$ (this is called the bounded certificate property);
(b) Given a (di)graph H, we can find in polynomial time either a spanning sub(di)graph H' of H such that $H' \in \Pi$ and $|A(H')| \leq b_\pi(|V(H)|)$, or a certificate showing that H contains no such subdigraph.

We formulate the following partitioning problem in terms of directed graphs but the same can be done for undirected graphs as well as mixed graphs.

$\{\Pi_1, \ldots, \Pi_\ell\}$-Arc-Disjoint Subgraphs Problem
Input: A directed graph D and a collection of families of good digraphs $\Pi_1, \ldots, \Pi_\ell$.
Question: Does D have subgraphs $D_1, \ldots, D_\ell$ such that

- for all $i \in \{1, \ldots, \ell\}$, $D_i \in \Pi_i$ and
- for all $1 \leq i < j \leq \ell$, $A(D_i) \cap A(D_j) = \emptyset$?

Next we outline an algorithm for $\{\Pi_1, \ldots, \Pi_\ell\}$-Arc-Disjoint Subgraphs Problem using the idea of random separation.

Theorem 14 *There is a one-sided-error Monte-Carlo algorithm for* $\{\Pi_1, \ldots, \Pi_\ell\}$-Arc-Disjoint Subgraphs Problem *running in* $\ell^t \cdot n^{\mathcal{O}(1)}$ *expected time. Here, n is the number of vertices in the input digraph and* $t = \sum_{i=1}^{\ell} b_{\pi_i}(n)$.

Proof The idea of the algorithm is similar to the algorithm in the proof of Theorem 7.

1. Uniformly at random color the arcs of D with $[\ell] = \{1, 2, \ldots, \ell\}$. Let the coloring function be called f.
2. Let H_i denote the subdigraph of D with the vertex set $V(D)$ and the arc set being those that have been assigned color i. In polynomial time decide whether each H_i contains a subdigraph D_i such that $D_i \in \Pi_i$ and $|A(D_i)| \leq b_{\pi_i}(n)$. If such a subdigraph exists for each $i \in [\ell]$ we return YES.

Let n be the number of vertices in the input digraph D. When we talk about a solution $(D_1, \ldots, D_\ell)$ to the $\{\Pi_1, \ldots, \Pi_\ell\}$-ARC-DISJOINT SUBGRAPHS PROBLEM we shall assume that each D_i satisfies that $|A(D_i)| \leq b_{\pi_i}(n)$. We can adjust any solution to satisfy this requirement since each Π_i is a good class of digraphs and thus satisfies the bounded certificate property. We say that the solution $(D_1, \ldots, D_\ell)$ to the $\{\Pi_1, \ldots, \Pi_\ell\}$-ARC-DISJOINT SUBGRAPHS PROBLEM is colorful with respect to f, if every arc of D_i has been colored with i. Since the number of arcs in D_i is bounded above by $b_{\pi_i}(n)$, the probability that a solution $(D_1, \ldots, D_\ell)$ becomes colorful is at least:

$$\prod_{i=1}^{\ell} \left(\frac{1}{\ell}\right)^{b_{\pi_i}(n)} = \ell^{-\left(\sum_{i=1}^{\ell} b_{\pi_i}(n)\right)} = \ell^{-t}.$$

Given a coloring, Step 2 of the algorithm finds a colorful solution satisfying the prescribed bounds on the arc sets if there exists one, in polynomial time. Thus, if there is indeed a solution $(D_1, \ldots, D_\ell)$ then we find it with probability ℓ^{-t}. So if we repeat the algorithm ℓ^t times, the error probability can be reduced to at most $1/e$, making the success probability at least $1 - \frac{1}{e} \geq \frac{1}{2}$. □

As a corollary of Theorem 14 we give efficient exact algorithms for the following problems.

(ℓ_1, ℓ_2)-ARC-DISJOINT IN-AND OUT-BRANCHINGS ((ℓ_1, ℓ_2)-ADB)
Input: A directed graph D on n vertices and m arcs.
Question: Does D have out-branchings $B^+_{s_1}, \ldots, B^+_{s_{\ell_1}}$ and in-branchings $B^-_{s_{\ell_1+1}}, \ldots, B^-_{s_{\ell_1+\ell_2}}$ such that they are pairwise disjoint.

This problem is NP-complete already when $\ell_1 = \ell_2 = 1$ [2].

NON DISCONNECTING SPANNING STRONG SUBDIGRAPH (NDSSS)
Input: A directed graph D on n vertices and m arcs.
Question: Does D have spanning tree T in D and a strong spanning subdigraph D' of D such that $A(D') \cap A(T) = \emptyset$?

This problem is NP-complete even for 2-regular digraphs [6].

Trivial brute-force algorithms for (ℓ_1, ℓ_2)-ADB and NDSSS work by guessing the arcs and then checking whether the arcs satisfy the desired properties. For an example; to solve NON DISCONNECTING SPANNING STRONG SUBDIGRAPH we can guess the arcs of D' and then check whether $UG(D^*)$ where D^* has vertex set $V(D)$ and arcs set $A(D) \setminus A(D')$ is connected. It is well known that we can obtain a strongly connected

digraph by taking an out-branching and an in-branching starting at a vertex s and thus the number of arcs in D' is at most $2n-2$. Also D' must contain at least n arcs and thus the overall running time of this algorithm will be

$$\sum_{i=n}^{2n-2} \binom{m}{i} \mathcal{O}(n+m) \sim \mathcal{O}(m^n).$$

However, we can apply the algorithm described in Theorem 14 and get an algorithm where the exponential part only depends on the number of vertices in the input graph and is independent of the number of arcs. Let Π_1 be class of all strongly connected digraphs, let Π_2 be the class of all trees and take $b_1(n) = 2n-2$ and $b_2(n) = n-1$. Given any strong digraph H on n vertices, we can find a spanning strong subdigraph on at most $b_1(n)$ arcs in linear time by fixing a vertex s and finding arbitrary in- and out-branchings B_s^-, B_s^+ rooted at s. The union of these forms H'. Clearly we can find a spanning tree an a given connected (di)graph in linear time. Hence by applying Theorem 14 we get the following corollary.

Corollary 15 NDSSS *can be solved in* $\mathcal{O}(8^n(n+m))$ *expected time.*

To solve (ℓ_1, ℓ_2)-Arc-Disjoint In-and Out-Branchings we proceed as follows. By the clasical theorem of Edmonds, given a digraph we can test whether it contains ℓ-arc disjoint (in-) out-branchings in polynomial time [15]. Thus, we fix Π_1 to be the class of digraphs that has ℓ_1 arc-disjoint out-branchings and Π_2 to be the class of digraphs that has ℓ_2 arc-disjoint in-branchings. We set $b_1(n) = \ell_1\,(n-1)$ and $b_2(n) = \ell_2\,(n-1)$. Now by applying Theorem 14 we get the following corollary.

Corollary 16 (ℓ_1, ℓ_2)-ADB *can be solved in* $\mathcal{O}(2^{(\ell_1+\ell_2)n}(n+m))$ *expected time.*

The algorithm in Theorem 14 can be derandomized using a generalization of universal sets which is defined as follows.

Definition 17 ([21]) An (n,k,q)-universal set is a family of vectors $V \subseteq [q]^n$ such that for any index set $S \in \binom{[n]}{k}$, the projection of V on S contains all q^k possible configurations.

Proposition 18 ([21]) *An* (n,k,q)*-universal family of cardinality* $q^k k^{\mathcal{O}(\log k)} \log^2 n$ *can be constructed deterministically in time* $\mathcal{O}(q^k k^{\mathcal{O}(\log k)} n \log^2 n)$.

Now using the (n,k,q)-universal family instead of random coloring in Theorem 14 we get the following result. In particular we apply Proposition 18 with the parameters $(|A(D)|, t, \ell)$.

Theorem 19 *There is an algorithm for* $\{\Pi_1, \ldots, \Pi_\ell\}$-Arc-Disjoint Subgraphs Problem *running in* $\ell^t \cdot n^{\mathcal{O}(1)}$. *Here, n is the number of vertices in the input digraph and* $t = \sum_{i=1}^{\ell} b_{\pi_i}(n)$.

One could obtain several exact algorithms as a direct corollary of Theorem 19. Recently Bernáth and Király [7] showed several natural packing, covering and partitioning problems to be NP-complete. For almost all the partitioning and covering problems mentioned in [7] one can obtain exact algorithms using Theorem 19. We leave the details, since these algorithms are direct consequences of our formulation of $\{\Pi_1, \ldots, \Pi_\ell\}$-Arc-Disjoint Subgraphs Problem.

6 Concluding remarks

In this paper we formulated some natural parameterized problems around finding a pair of branchings or spanning trees differing on at least k arcs and obtained some parameterized algorithms for them. We conclude with presenting the following questions, whose parameterized complexity status is still unknown.

Problem 1 *Given a strong digraph $D = (V, A)$ and a natural number k. Does D contain a spanning tree T such that we can delete some set of k arcs of $A(T)$ from D and still have a strong spanning subdigraph of D?*

A related, seemingly simpler question is the following.

Problem 2 *Given a digraph $D = (V, A)$ and a number k. Can we delete some set X of k arcs from D such that $D\backslash X$ is strong?*

Acknowledgments We would like to thank the reviewers for their valuable comments and suggestions.

References

1. Alon, N., Fomin, F.V., Gutin, G., Krivelevich, M., Saurabh, S.: Spanning directed trees with many leaves. SIAM J. Discrete Math. **23**(1), 466–476 (2009)
2. Bang-Jensen, J.: Edge-disjoint in- and out-branchings in tournaments and related path problems. J. Combin. Theory Ser. B **51**(1), 1–23 (1991)
3. Bang-Jensen, J., Gutin, G.: Digraphs: Theory, Algorithms and Applications, 2nd edn. Springer, London (2009)
4. Bang-Jensen, J., Simonsen, S.: Arc-disjoint paths and trees in 2-regular digraphs. Discrete Appl. Math. **161**(1617), 2724–2730 (2013)
5. Bang-Jensen, J., Yeo, A.: The minimum spanning subdigraph problem is fixed parameter tractable. Discrete Appl. Math. **156**, 2924–2929 (2008)
6. Bang-Jensen, J., Yeo, A.: Arc-disjoint spanning subdigraphs in digraphs. Theor. Comput. Sci. **438**, 48–54 (2012)
7. Bernáth, A., Király, Z.: On the tractability of some natural packing, covering and partitioning problems. Discrete Appl. Math. **180**, 25–35 (2015)
8. 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 Trans. Algorithms **8**(4), 38 (2012)
9. Binkele-Raible, D., Fernau, H., Gaspers, S., Liedloff, M.: Exact and parameterized algorithms for max internal spanning tree. Algorithmica **65**(1), 95–128 (2013)
10. Cohen, N., Fomin, F.V., Gutin, G., Kim, E.J., Saurabh, S., Yeo, A.: Algorithm for finding k-vertex out-trees and its application to k-internal out-branching problem. J. Comput. Syst. Sci. **76**(7), 650–662 (2010)
11. Courcelle, B.: The monadic second-order logic of graphs. I: recognizable sets of finite graphs. Inf. Comput. **85**(1), 12–75 (1990)
12. Courcelle, B.: The monadic second-order logic of graphs. III: tree-decompositions, minor and complexity issues. ITA **26**, 257–286 (1992)
13. Daligault, J., Gutin, G., Kim, E.J., Yeo, A.: Fpt algorithms and kernels for the directed k-leaf problem. J. Comput. Syst. Sci. **76**(2), 144–152 (2010)
14. Dorn, F., Fomin, F.V., Lokshtanov, D., Raman, V., Saurabh, S.: Beyond bidimensionality: parameterized subexponential algorithms on directed graphs. STACS **5**, 251–262 (2010)
15. Edmonds J.: Matroid partition. In: Dantzig G.B., Veinott A.F. (eds.) Mathematics of the Decision Sciences: Part 1, pp. 335–345. American Mathematical Society, New York (1968)
16. Edmonds., J.: Combinatorial Algorithms. In: Rustin, B. (ed.) Edge-disjoint branchings, pp. 91–96. Academic Press, Waltham (1973)
17. Fomin, F., Kratsch, D.: Exact Exponential Algorithms. Springer, Berlin (2010)

18. Fomin, F.V., Gaspers, S., Saurabh, S., Thomassé, S.: A linear vertex kernel for maximum internal spanning tree. J. Comput. Syst. Sci. **79**(1), 1–6 (2013)
19. Fomin, F.V., Grandoni, F., Lokshtanov, D., Saurabh, S.: Sharp separation and applications to exact and parameterized algorithms. Algorithmica **63**(3), 692–706 (2012)
20. Fomin, F.V., Lokshtanov, D., Saurabh, S.: Efficient computation of representative sets with applications in parameterized and exact algorithms. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 142–151 (2013)
21. Neeldhara, M., Fahad, P., Rai, A., Raman, V., Saurabh, S.: Parameterized algorithms for max colorable induced subgraph problem on perfect graphs. In *Graph-Theoretic Concepts in Computer Science—39th International Workshop, WG 2013, Lübeck, Germany, June 19–21, 2013, Revised Papers, vol. 8165 of Lecture Notes in Computer Science*, pp. 370–381. Springer, Berlin (2013)
22. Naor, M., Schulman, L.J., Srinivasan, A.: Splitters and near-optimal derandomization. (Preliminary version). In 36th Annual Symposium on Foundations of Computer Science. Held in Milwaukee, WI, USA, October 23–25, 1995. Los Alamitos, CA: IEEE Computer Society Press. pp. 182–193 (1995)