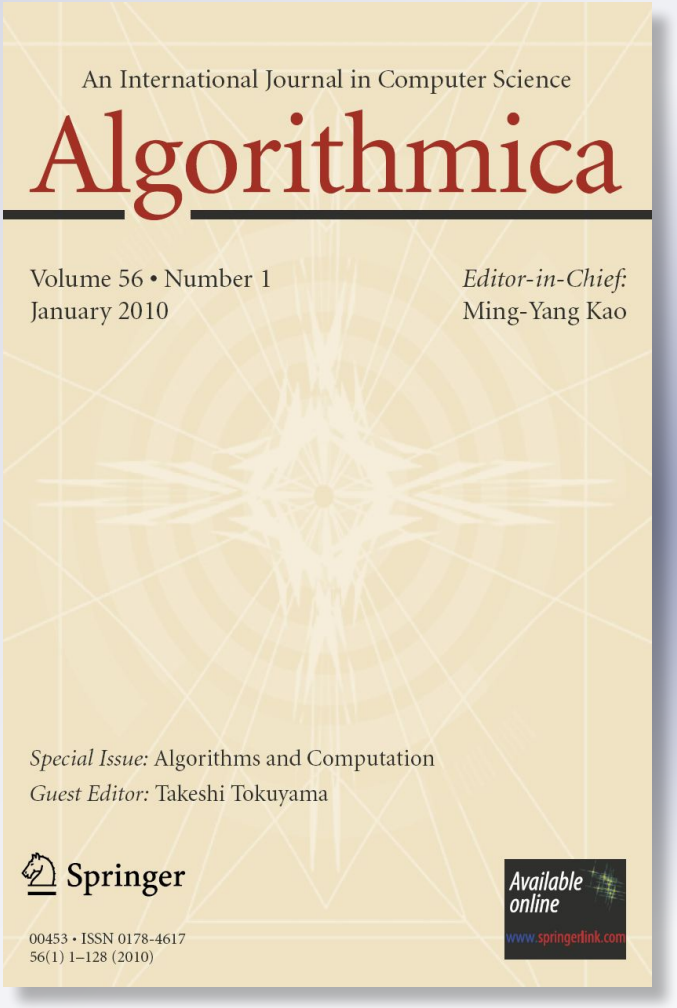
Sharp Separation and Applications to Exact and Parameterized Algorithms

Fedor V. Fomin, Fabrizio Grandoni, Daniel Lokshtanov & Saket Saurabh

Algorithmica

ISSN 0178-4617

Algorithmica
DOI 10.1007/s00453-011-9555-9



Springer

Algorithmica
DOI 10.1007/s00453-011-9555-9

Sharp Separation and Applications to Exact and Parameterized Algorithms

Fedor V. Fomin · Fabrizio Grandoni · Daniel Lokshtanov · Saket Saurabh

Received: 18 August 2010 / Accepted: 19 July 2011

Abstract Many divide-and-conquer algorithms employ the fact that the vertex set of a graph of bounded treewidth can be separated in two roughly balanced subsets by removing a small subset of vertices, referred to as a *separator*. In this paper we prove a trade-off between the size of the separator and the sharpness with which we can fix the size of the two sides of the partition. Our result appears to be a handy and powerful tool for the design of exact and parameterized algorithms for NP-hard problems. We illustrate that by presenting two applications.

Our first application is a $O(2^{n+o(n)})$-time algorithm for the DEGREE CONSTRAINED SPANNING TREE problem: find a spanning tree of a graph with the maximum number of nodes satisfying given degree constraints. This problem generalizes some well-studied problems, among them Hamiltonian Path, Full Degree Spanning Tree, Bounded Degree Spanning Tree, and Maximum Internal Spanning Tree.

The second application is a parameterized algorithm with running time $O(16^{k+o(k)} + n^{O(1)})$ for the k-INTERNAL OUT-BRANCHING problem: here the goal is to compute an out-branching of a digraph with at least k internal nodes. This is

A preliminary version of this paper appeared in the proceedings of LATIN 2010 [20].

F.V. Fomin · D. Lokshtanov
Department of Informatics, University of Bergen, Bergen, Norway

F.V. Fomin
e-mail: fedor.fomin@ii.uib.no

D. Lokshtanov
e-mail: daniello@ii.uib.no

F. Grandoni
Computer Science Department, University of Rome Tor Vergata, Roma, Italy
e-mail: grandoni@disp.uniroma2.it

S. Saurabh (✉)
The Institute of Mathematical Sciences, Chennai, India
e-mail: saket@imsc.res.in

Published online: 13 August 2011

a significant improvement over the best previously known parameterized algorithm for the problem by Cohen et al. (J. Comput. Syst. Sci. 76:650–662, 2010), running in time $O(49.4^k + n^{O(1)})$.

1 Introduction

The aim of *parameterized* and *exact* algorithms is to solve NP-hard problems exactly, with the smallest possible (exponential) worst-case running time. While exact algorithms are designed to minimize the running time as a function of the input size, parameterized algorithms seek to perform better when the instance considered has more structure than a general instance to the problem. (For an introduction to the topic, see e.g. [13, 40]). Exact and parameterized algorithms have an old history [9, 27, 33], but they have been at the forefront in the last decade. In the last few years, many new techniques have been developed to design and analyze exact algorithms, among them Inclusion-Exclusion [4], Subset Convolution [5], Measure & Conquer [16], and Iterative Compression [43]. Among the best-studied problems, let us recall Independent Set [7, 16, 44, 46], (Connected) Dominating Set [14, 16, 17, 25, 47], Steiner Tree [5, 15, 21, 39], Feedback Vertex Set [18, 42], Coloring [2, 4, 33], Satisfiability [32, 37], Traveling Salesman and Hamiltonian Path [3, 27, 28, 31, 36, 48], and many others.

A classical approach to solve combinatorial problems is *divide-and-conquer*: decompose the problem in two or more sub-problems, solve them independently and merge the obtained solutions. Several divide-and-conquer algorithms rely on the existence of a small *separator*, which is defined as follows. Let G be an n-vertex graph with vertex set $V = V(G)$ and edge set $E = E(G)$. A set of vertices S is called an α-separator of G, $0 < \alpha \leq 1$, if the vertex set $V \setminus S$ can be partitioned into sets V_L and V_R of size at most αn such that no vertex of V_L is adjacent to any vertex of V_R. For example, the classical result of Lipton and Tarjan [34] that every planar graph has a $\frac{2}{3}$-separator of size $O(\sqrt{n})$ can be used to solve many NP-hard problems in planar graphs in time $O(2^{O(\sqrt{n})})$ [35]. Alon, Seymour and Thomas [1] generalized the result of Lipton and Tarjan [34] and proved that every graph excluding a fixed graph H as a minor has a $\frac{2}{3}$-separator of size $O(|V(H)|^{3/2}\sqrt{n})$. It is well known that every tree has a $\frac{2}{3}$-separator of size 1. This result was generalized to graphs of bounded treewidth in [6], where it is shown that every graph of treewidth t has a $\frac{2}{3}$-separator of size at most $t + 1$.

1.1 Our Results

In this paper (see Sect. 2) we prove a trade-off between the size of the separator S and the sharpness with which we can fix the size of V_L and V_R in the partition, for graphs of treewidth t (see Sect. 1.2 for the definition of treewidth). Given a function $w : X \to \mathbb{R}$, we define $w(Y) = \sum_{y \in Y} w(y)$ for any $Y \subseteq X$.

Lemma 1 (Sharp Separation) *Let $G = (V, E)$ be a graph of treewidth t and $w : V \to \{0, 1\}$. Then for any integer $p \geq 0$ and $0 \leq x \leq w(V)$, there is a partition (V_L, S, V_R) of V such that $|S| \leq t \cdot p$, $w(V_L) \leq x + \lfloor \frac{w(V)}{2^p} \rfloor$, $w(V_R) \leq w(V) -$*

$x + \lfloor \frac{w(V)}{2^p} \rfloor$, and there is no edge in G with one endpoint in V_L and the other endpoint in V_R, that is, S separates V_L from V_R. Given a tree-decomposition of G of width t, S can be computed in polynomial time.

Here the weight function w is used to model a subset $W \subseteq V$ of vertices that we wish to separate. Lemma 1 implies for example that, with a separator of logarithmic size (for bounded treewidth graphs), we can obtain a *perfectly balanced* partition with $\max\{|V_L|, |V_R|\} \leq n/2$. In this paper we will always choose $p = \log_{2-\varepsilon} n$, for a small constant $\varepsilon > 0$, so that the additive term $\lfloor \frac{w(V)}{2^p} \rfloor$ disappears. The base of the logarithm will be omitted, in order to lighten the notation.

Let us remark that, in the applications considered in this paper, we will use a restricted version of the above lemma where the considered graph is a tree (hence $t = 1$), and all node weights are one. Furthermore, we will never need to compute a tree-decomposition (since for our algorithms the only thing we need is the separator and that will be *guessed*). However, we decided to present the lemma in the above form since it captures other applications which are not discussed here. For example, w can be used to model Steiner-tree-like problems. Furthermore, our basic approach (differently from other techniques in the literature) allows one to find graphs of bounded treewidth (with given properties) rather than trees.

Our Sharp Separation Lemma is a handy tool in the design of parameterized and exact algorithms based on the divide-and-conquer paradigm. We illustrate that by presenting two applications.

Degree Constrained Spanning Tree For a given graph $G = (V, E)$, let $d_G(v)$ denote the degree of $v \in V$ in G. Our first result is an algorithm for the following problem:

DEGREE CONSTRAINED SPANNING TREE (DCST). Given a graph $G = (V, E)$ and a function $\mathcal{D} : V \to 2^{\{1,\ldots,n\}}$. Find a spanning tree T of G maximizing $|\{v \in V : d_T(v) \in \mathcal{D}(v)\}|$.

Intuitively, $\mathcal{D}(v)$ can be seen as a set of desirable degrees for a vertex v in the spanning tree. We have a *hit* each time $d_T(v) \in \mathcal{D}(v)$ for some v. The goal is maximizing the number of hits.

DCST naturally generalizes many NP-hard spanning tree and path problems studied in the literature. For instance we can code the famous HAMILTONIAN PATH problem, find a spanning path of a given graph, by letting $\mathcal{D}(v) = \{1, 2\}$ for all vertices; A spanning tree with n hits is a Hamiltonian path. Another example is the FULL DEGREE SPANNING TREE problem, where we search for a spanning tree which maximizes the number of vertices having the same degree in the graph as in the tree [29]. To code this problem we set $\mathcal{D}(v) = \{d_G(v)\}$ for every vertex v. Another well-studied spanning tree problem is the BOUNDED DEGREE SPANNING TREE problem [22, 24, 45]. Here we search for a spanning tree such that the degree of each vertex v in the spanning tree is bounded by an integer $B_v \leq n - 1$ given as input. Clearly setting $\mathcal{D}(v) = \{1, \ldots, B_v\}$ yields an encoding of this problem as well. Finally, to code the MAXIMUM INTERNAL SPANNING TREE problem, where the aim is to find a spanning tree maximizing the number of internal vertices, we set $\mathcal{D}(v) = \{2, \ldots, d_G(v)\}$ for every vertex v.

One of the earliest results in the field of exact algorithms [27, 28, 31] is a $O(2^n n^{O(1)})$ time algorithm for HAMILTONIAN PATH. In a very recent, breakthrough paper, Björklund [3] improved the running time to $O(1.66^n)$. Gaspers et al. [23] give a $O(1.92^n)$ time algorithm for FULL DEGREE SPANNING TREE. Fernau et al. [12] give a $O(3^n n^{O(1)})$ time algorithm for MAXIMUM INTERNAL SPANNING TREE [12].

Here we present an algorithm which solves DCST in $O(2^{n+o(n)})$ time and space, where n is the number of nodes in the graph. Recently and independently, Nederlof [39] gave an Inclusion-Exclusion based algorithm running in time $O(2^n n^{O(1)})$ and polynomial space for DCST. Though his technique gives better results for DCST, our approach seems to be more flexible. In particular, our method works for the weighted version of the problem (with arbitrarily large weights).

k-Internal Out-Branching The second application of the Sharp Separation Lemma is a parameterized algorithm for the following problem.

k-INTERNAL OUT-BRANCHING: Given a digraph $G = (V, E)$ and a positive integer k, check whether there exists an out-branching with at least k internal vertices.

The *undirected* counterpart to this problem, namely k-INTERNAL SPANNING TREE, was first studied by Prieto and Sloper [41], who gave an algorithm with running time $2^{4k \log k} n^{O(1)}$ and a kernel of size $O(k^2)$ for the problem. Recently, Fomin et al. [19] gave an improved algorithm with running time $8^k n^{O(1)}$ and a kernel with at most $3k$ vertices. For k-INTERNAL OUT-BRANCHING, Gutin et al. [26] obtained an algorithm of running time $2^{O(k \log k)} n^{O(1)}$ and gave a kernel of size $O(k^2)$. A faster algorithm, running in time $O(49.4^k n^{O(1)})$ was subsequently developed by Cohen et al. [10]. In this paper we use the Sharp Separation Lemma to obtain an algorithm with running time $O(16^{k+o(k)} + n^{O(1)})$. The space complexity is exponential (in k). However, it can be made polynomial by means of randomization.

1.2 Preliminaries

For basic graph terminology we refer the reader, e.g., to [11]. Let $G = (V, E)$ be an undirected graph, $V' \subseteq V$ and $E' \subseteq E$. The degree of node v is denoted by $d_G(v)$. By $G[V']$ we denote the subgraph of G induced by V'. We use $G - V'$ as a shortcut for $G[V - V']$. We also use the shortcut $G - E' = (V, E - E')$. We sometimes confuse E' with the corresponding subgraph. For a subgraph G', $V(G')$ and $E(G')$ denote the node and edge set of G', respectively. For two subgraphs G' and G'', $G' \cup G'' = (V(G') \cup V(G''), E(G') \cup E(G''))$. We use $G' \uplus G'' = (V(G') \cup V(G''), E(G') \uplus E(G''))$ to denote the corresponding multi-graph, where edges are counted with their multiplicity. By *contracting* an undirected edge $\{u, v\}$, we mean replacing u and v with a new node z, which inherits all the edges incident to u and v. Symmetrically, by *splitting* $\{u, v\}$, we mean replacing it with two edges $\{u, z\}$ and $\{z, v\}$, where z is a newly created node.

Consider now a digraph $G = (V, E)$. We use the same notation as above, with analogous meaning. Let us remark that, when splitting (u, v), the two new edges are (u, z) and (z, v). Furthermore, we let $d^+(v)$ be the out-degree of node v. An *r-out-tree* in a digraph G is a subtree T of G rooted at r, such that all arcs of T are oriented away from r. If T contains all vertices of G, T is said to be an *r-out-branching*.

A *tree decomposition* of a (undirected) graph $G = (V, E)$ is a pair (X, U) where $U = (W, F)$ is a tree, and $X = (\{X_i \mid i \in W\})$ is a collection of subsets of V such that: (i) $\bigcup_{i \in W} X_i = V$, (ii) for each edge $\{v, u\} \in E$, there is an $i \in W$ such that $v, u \in X_i$, and (iii) for each $v \in V$ the set of vertices $\{i \mid v \in X_i\}$ forms a subtree of U. The *width* of (X, U) is $\max_{i \in W}\{|X_i| - 1\}$. The *treewidth* $tw(G)$ of G is the minimum width over all the tree decompositions of G. We recall that the treewidth of a tree is 1.

We will exploit the following definition and theorem.

Definition 2 (See [38]) An (n, t)-universal set $\mathcal{F}$ is a set of functions from $\{1, \ldots, n\}$ to $\{0, 1\}$, such that for every subset $S \subseteq \{1, \ldots, n\}$, $|S| = t$, the set $\mathcal{F}|_S = \{f|_S \mid f \in \mathcal{F}\}$ is equal to the set 2^S of all the functions from S to $\{0, 1\}$.

Theorem 3 (See [38]) *There is a deterministic algorithm with running time* $O(2^t t^{O(\log t)} n \log n)$ *that constructs an* (n, t)*-universal set* $\mathcal{F}$ *such that* $|\mathcal{F}| = 2^t t^{O(\log t)} \log n$.

2 Sharp Separation in Graphs of Bounded Treewidth

In this section we prove our Sharp Separation Lemma, which is at the heart of the algorithms described in the following sections. In order to prove that, we need the following well-known result.

Lemma 4 (See [6]) *Given a* n*-vertex graph* $G = (V, E)$ *of treewidth* t *and* $w : V \to \{0, 1\}$. *There is a set* T *of vertices of size at most* $t + 1$ *such that for any connected component* $G[C]$ *of* $G \setminus T$, $w(C) \leq w(V)/2$. *Given a tree-decomposition of* G *of width* t, T *can be computed in polynomial time.*

Now we are ready to prove Lemma 1.

Proof of Lemma 1 We construct V_L, V_R and S iteratively, starting from empty sets, as follows. By Lemma 4 there is a set T of size at most t such that for any connected component $G[C]$ of $G \setminus T$, $w(C) \leq w(V)/2$. We add T to S and for each component $G[C]$ of $G \setminus T$, add C to V_L or V_R if this does not violate $w(V_L) \leq x$ or $w(V_R) \leq w(V) - x$, respectively.

Let us show that at the end of the process there is at most one component $G[C]$ left. Suppose by contradiction that there are at least 2 such components, say $G[C_1]$ and $G[C_2]$. Without loss of generality assume $w(C_1) \leq w(C_2)$. This implies that $w(V_L) + w(C_1) > x$ and $w(V_R) + w(C_1) > w(V) - x$. Consequently,

$$w(V_L) + w(V_R) + 2w(C_1) > w(V).$$

However, this contradicts the fact that

$$w(V_L) + w(V_R) + 2w(C_1) \leq w(V_L) + w(V_R) + w(C_1) + w(C_2) \leq w(V).$$

Now we iteratively reapply the construction above for $p-1$ times, each time considering the component $G[C]$ left from previous step. Eventually we add C to either V_L or V_R.

At each iteration the weight of the considered component C halves, so at the end of the process $w(C) \leq \lfloor w(V)/2^p \rfloor$. The upper bound on the weight of V_L and V_R follows. Since at each step we add to S a set of size at most t, we eventually obtain $|S| \leq t \cdot p$. The running time claim follows immediately from Lemma 4. This concludes the proof. □

3 Degree Constrained Spanning Tree

In this section we present our $O(2^{n+o(n)})$-time algorithm for the DEGREE CONSTRAINED SPANNING TREE problem (DCST). Indeed, we rather consider a weighted generalization of the problem. Here we are given an undirected graph $G = (V, E)$, with node weights $w : V \to \mathbb{R}_{\geq 0}$, and a list of *desirable* degrees $\mathcal{D}(v)$ for each vertex v. The hits $\text{hit}(G')$ of a subgraph G' is the set of nodes v such that $d_{G'}(v) \in \mathcal{D}(v)$. Our goal is to find a spanning tree T of maximum weight $w(\text{hit}(T)) := \sum_{v \in \text{hit}(T)} w(v)$.

Our recursive algorithm is described in Fig. 1. The base case of the recursion is given in Step (1). If there is no solution, the algorithm returns the empty graph $\emptyset$. Here the algorithm exploits an (initially empty) memoization table, where it stores the solutions to each solved subproblem. This prevents the algorithm from solving the same subproblem twice. There is a technical detail which is worth discussing. Due to the creation of new nodes, the subgraphs created by the algorithm are not induced subgraphs of the initial graph. This creates some troubles when one needs to search for a subproblem in the memoization table. One way to solve this issue is to guarantee that original nodes maintain the same labels in the subproblems as in the

dcst$(G, w, \mathcal{D})$

(1) If G is disconnected, return $\emptyset$. If $M = 0$, return any spanning tree. If $n \leq a$, solve the problem by brute force and return the obtained solution. If the problem solution is present in the memoization table, return it.
(2) For any subset of nodes $S \subseteq V(G)$, with $|S| \leq \log n$, for any two disjoint subsets $E_L, E_R \subseteq S^2 (= S \times S)$ such that $E_L \cup E_R$ is a spanning tree of (S, S^2), for any bipartion (V_L, V_R) of $V - S$ such that $|V_L| \leq n/\log^2 n$ and $|V_R| \leq n - n/\log^2 n$, for any two degree assignments $d_L : S \to \{1, \ldots, n-1\}$ and $d_R : S \to \{1, \ldots, n-1\}$:
  (2.a) Construct a graph G_L from G, by removing nodes V_R, adding edges E_R, and splitting those edges. Let D_R and F_R be the new nodes and edges, respectively, created by the splittings.
  (2.b) Define a node weight function w_L on G_L, with $w_L(v) = M' := M \cdot (n+1)$ for $v \in D_R \cup S$, and $w_L(v) = w(v)$ otherwise.
  (2.c) Define degree constraints $\mathcal{D}_L$ on G_L, with $\mathcal{D}_L(v) = \{d_L(v)\}$ for $v \in S$, $\mathcal{D}_L(v) = \{2\}$ for $v \in D_R$, and $\mathcal{D}_L(v) = \mathcal{D}(v)$ otherwise.
  (2.d) Compute $SOL'_L := \text{dcst}(G_L, w_L, \mathcal{D}_L)$, and let $SOL_L := SOL'_L - D_R$.
  (2.e) Compute SOL_R symmetrically. Let $SOL := SOL_L \cup SOL_R$.
(3) Among the subgraphs SOL computed above, return a feasible solution of maximum weight, if any. Otherwise, return $\emptyset$.

Fig. 1 Algorithm for DCST. Here a is a sufficiently large constant, M is the largest input node weight, and n is the number of nodes

input problem, while new nodes take different labels. As we will discuss, the number of new nodes at any time is bounded by a poly-logarithm in the initial number of nodes: hence one can find the desired entry in the table in sub-exponential time $2^{o(n)}$.

In Step (2) the algorithm creates a set of pairs of DCST instances $(G_L, w_L, \mathcal{D}_L)$ and $(G_R, w_R, \mathcal{D}_R)$, and solves them recursively. The reasons behind the choice of those pairs will be clearer from the correctness analysis. In the subproblems we set the weight of some nodes to a very large value, and restrict their degree set to a unique value. Intuitively, this forces the corresponding solutions to set the degree of those nodes to the mentioned values.

Lemma 5 (Correctness) *If there exists a feasible solution, algorithm* dcst *returns one such solution of maximum weight.*

Proof By definition, if the algorithm returns a solution, it is feasible. Let us prove by induction on n that, if there exists a feasible solution, the algorithm returns one such solution of maximum weight. The claim is trivially true if the algorithm halts at Step (1).

Otherwise, consider the following choice for the tuple $(S, E_L, E_R, V_L, V_R, d_L, d_R)$ (see also Fig. 2a), with the corresponding graphs SOL, SOL_L etc. Let OPT be the optimal solution. We let S be a minimum-cardinality separator of OPT which partitions $V - S$ into (V_L, V_R) as required. By the Sharp Separation Lemma, $|S| \leq \log n$ as needed. Define $OPT_L := OPT[V_L \cup S]$ and $OPT_R := OPT[V_R \cup S] - E(OPT[S])$. Observe that OPT_L and OPT_R bipartition the edges of OPT.

We next define E_R and d_L, the definition of E_L and d_R being symmetric. Let us iteratively contract the edges of OPT_R which contain at least one node outside S (the new node inherits the label of the endpoint in S, if any). The resulting set of edges in S^2 defines E_R. Let us remark that $E_L \cup E_R$ defines a spanning tree on node set S. For

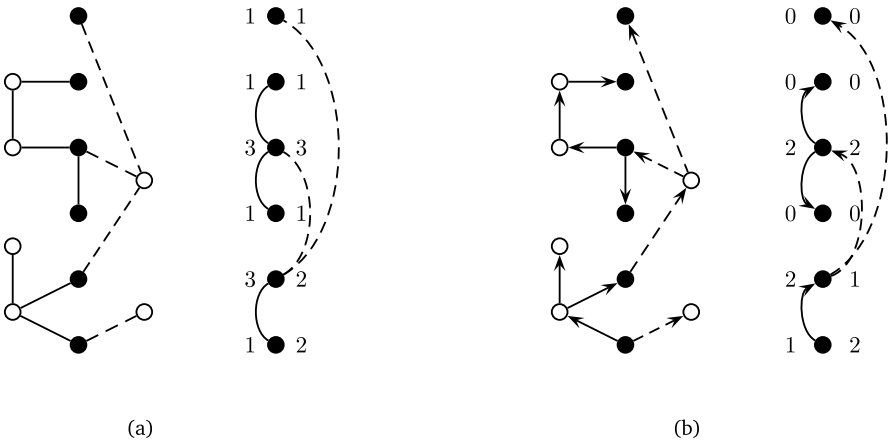


Fig. 2 (**a**) Example of construction of (E_L, E_R, d_L, d_R) for DCST. *On the left*, the optimum solution OPT (dashed edges belong to OPT_R). Black nodes belong to S. *On the right*, the edges E_L (full) and E_R (*dashed*). The degrees d_L and d_R are indicated at *the left* and *right* of each node, respectively. (**b**) Example of construction of (E_L, E_R, d_L^+, d_R^+) for DCOT, with an analogous notation

any $s \in S$, we set $d_L(s) := d_{OPT_L}(s) + d_{E_R}(s)$ (i.e., the degree of s in $OPT_L \cup E_R$, since OPT_L and E_R are edge-disjoint).

Let us first show that $SOL = SOL_L \cup SOL_R$ is a spanning tree, i.e. for any two distinct nodes s' and s'', there exists exactly one (simple) path among them in SOL. We prove the stronger claim that $SOL' := SOL_L \uplus SOL_R$ satisfies that property. By construction, it is sufficient to prove the claim for $s', s'' \in S$. Both $SOL'_L = SOL_L \cup F_R = SOL_L \uplus F_R$ and $SOL'_R = SOL_R \cup F_L = SOL_R \uplus F_L$ contain exactly one path between s' and s''. Hence $SOL'_L \uplus SOL'_R = (SOL_L \uplus SOL_R) \uplus (F_L \uplus F_R) = SOL' \uplus (F_L \cup F_R)$ contains exactly two edge disjoint paths between s' and s''. Since $(F_L \cup F_R)$ contains exactly one such path (being $E_L \cup E_R$ a tree), the same must hold for SOL'.

It remains to show that $w(\text{hit}(SOL)) \geq w(\text{hit}(OPT))$. Observe that $OPT'_L := OPT_L \cup F_R$ is a feasible solution to subproblem $(G_L, w_L, \mathcal{D}_L)$. By construction

$$\text{hit}(OPT'_L) = (\text{hit}(OPT'_L) \cap V_L) \cup (D_R \cup S) = (\text{hit}(OPT_L) \cap V_L) \cup (D_R \cup S),$$

which implies $w_L(\text{hit}(OPT'_L)) \geq M'|D_R \cup S|$. As a consequence,

$$\text{hit}(SOL'_L) = (\text{hit}(SOL'_L) \cap V_L) \cup (D_R \cup S) = (\text{hit}(SOL_L) \cap V_L) \cup (D_R \cup S).$$

In fact, any solution not satisfying this property would have weight at most $M'(|D_R \cup S| - 1) + Mn = M'|D_R \cup S| - M < w_L(\text{hit}(OPT'_L))$. We can conclude that

$$\begin{aligned} w(\text{hit}(SOL_L) \cap V_L) &= w_L(\text{hit}(SOL_L) \cap V_L) = w_L(\text{hit}(SOL'_L)) - w_L(D_R \cup S) \\ &\geq w_L(\text{hit}(OPT'_L)) - w_L(D_R \cup S) = w_L(\text{hit}(OPT_L) \cap V_L) \\ &= w(\text{hit}(OPT_L) \cap V_L). \end{aligned} \tag{1}$$

A symmetric argument yields

$$w(\text{hit}(SOL_R) \cap V_R) \geq w(\text{hit}(OPT_R) \cap V_R). \tag{2}$$

Observe that F_R is contained in both SOL'_L and OPT'_L. Hence any node $v \in S$ has the same degree in SOL_L and OPT_L. Similarly for SOL_R and OPT_R. Consequently:

$$H := \text{hit}(OPT) \cap S = \text{hit}(SOL) \cap S. \tag{3}$$

Putting everything together:

$$\begin{aligned} w(\text{hit}(SOL)) &= w(\text{hit}(SOL) \cap V_L) + w(\text{hit}(SOL) \cap V_R) + w(\text{hit}(SOL) \cap S) \\ &\overset{(3)}{=} w(\text{hit}(SOL_L) \cap V_L) + w(\text{hit}(SOL_R) \cap V_R) + w(H) \\ &\overset{(1)+(2)}{\geq} w(\text{hit}(OPT_L) \cap V_L) + w(\text{hit}(OPT_R) \cap V_R) + w(H) \\ &\overset{(3)}{=} w(\text{hit}(OPT)). \end{aligned}$$

□

Lemma 6 (Running time) *The running time and space complexity of algorithm* dcst *is* $O(2^{n+o(n)})$.

Proof The space complexity of the algorithm is upper bounded by the overall cost of constructing the memoization table, and hence by the running time of the algorithm. Thus it is sufficient to bound the running time.

This time is bounded, modulo a sub-exponential factor $2^{o(n)}$, by the number of possible entries in the memoization table, times the number of possible tuples $(S, E_L, E_R, V_L, V_R, d_L, d_R)$. The latter quantity is at most:

$$2\binom{n}{\log n} \cdot \binom{\log^2 n}{\log n - 1} 2^{\log n - 1} \cdot 2\binom{n}{n/\log^2 n} \cdot n^{\log n} \cdot n^{\log n} = 2^{o(n)}.$$

In order to bound the number of table entries, observe that at each recursive call the number of nodes in each subproblem decreases by a factor $(1 - 1/\log^2 n)$ or more. Consequently, the depth of the recursion tree is $O(\log^3 n)$. Since the new nodes created at each recursive call are $O(\log n)$, the number of new nodes with respect to the initial problem is $O(\log^4 n)$ in any subproblem. By the same argument, the overall number of nodes involved in some set S (including ancestor subproblems) is $O(\log^4 n)$ in any subproblem. Hence any graph in some subproblem can be obtained via the following procedure: take the subgraph induced by a subset of nodes $V' \subseteq V$, add $O(\log^4 n)$ new nodes W, and add arbitrary edges $\{u, v\}$ with at least one endpoint in W. The number of graphs which can be obtained with this procedure is $2^n \cdot O(\log^8 n) \cdot O(n \cdot \log^4 n) = 2^{n+o(n)}$.

Node weights are either original weights or weights of the type $(M \cdot (n+1))^i$, with $i = O(\log^3 n)$ and M the largest weight in the initial instance. The latter case can happen only for $O(\log^4 n)$ nodes. As a consequence, the possible weight functions for a given graph are at most

$$\binom{n}{O(\log^4 n)} O((\log^3 n)^{O(\log^4 n)}) = 2^{o(n)}.$$

By a similar argument, the possible degree constraint functions on a given graph are

$$\binom{n}{O(\log^4 n)} O(n^{O(\log^4 n)}) = 2^{o(n)}.$$

Hence the memoization table has size $2^{n+o(n)}$. The claim follows. □

Lemmas 5 and 6 together imply the following theorem.

Theorem 7 *There is a $2^{n+o(n)}$ time and space algorithm for (node weighted) DCST.*

Remark The approach above can be adapted (at the cost of a more technical algorithm and analysis) to find degree constrained spanning subgraphs of treewidth t in time $O(2^{n+o(n)})$ for every fixed constant t. The basic idea is exploiting separators of size $O(t \log n)$, whose existence is guaranteed by the Sharp Separation Lemma.

4 k-Internal Out-Branching

In this section we give a parameterized algorithm with running time $O(16^{k+o(k)} + n^{O(1)})$ for the k-INTERNAL OUT-BRANCHING problem. Our approach combines the Sharp Separation Lemma with the *divide-and-color* paradigm in [8, 30], and a polynomial-size kernel for the problem [26]. One important difference with respect to the algorithm in previous section is that the Sharp Separation Lemma is used to divide the problem into balanced (rather than very unbalanced) subproblems.

4.1 Some Reductions

The first step of our algorithm is to apply the *kernelization algorithm* of Gutin et al. [26]. Given an instance (G, k) of k-INTERNAL OUT-BRANCHING, that algorithm produces a new instance (G', k') with $|V(G')| = O(k^2)$ and $k' \leq k$ such that G' has an out-branching with at least k' internal vertices if and only if G has an out-branching with at least k internal vertices. After this step we can assume that the number $\overline{n}$ of vertices in the input digraph G is $O(k^2)$.

Now, the algorithm guesses the root r of the out-branching (by trying all the $\overline{n}$ possible values), and verifies that there indeed is some out-branching of G rooted at r. This guessing step, together with the following observation, allows us to search for r-out-trees instead of r-out-branchings of G.

Lemma 8 (See [10]) *Let G be a digraph and r be a node of G such that there is an r-out-branching of G. Then, for any r-out-tree T with at least k internal nodes there is an r-out-branching T' with at least k internal nodes containing T as a subtree. Moreover such a T' can be found in polynomial time.*

When looking for r-out-trees with at least k internal nodes, it is sufficient to restrict ourselves to r-out-trees with at most $2k$ nodes. The reason for this is that if some internal node sees at least two leaves of the r-out-tree, then one of the leaves can be removed without changing any internal node into a leaf. We formalize this as an observation.

Lemma 9 (See [10]) *Let G be a digraph and r be a node of G. If there is an r-out-tree T with at least k internal nodes then there is an r-out-tree T' on at most $2k$ nodes with at least k internal nodes.*

At this point it is convenient to turn the original problem into a weighted maximization problem, with degree constraints. This way, we can re-use most of the machinery developed for DCST. In more detail, let us consider the following DEGREE-CONSTRAINED OUT-TREE problem (DCOT). We are given a directed graph $G = (V, E)$, with node weights $w : V \to \mathbb{R}_{\geq 0}$ and out-degree sets $\mathcal{D}^+(v) \subseteq \{0, \ldots, n-1\}$ for every $v \in V$. For a subgraph G', the hits $\text{hit}(G')$ of G' is the set of nodes v such that $d^+_{G'}(v) \in \mathcal{D}^+(v)$. The goal is finding an r-out-tree T on at most t nodes which maximizes $w(\text{hit}(T)) := \sum_{v \in \text{hit}(T)} w(v)$.

The original problem can be encoded in a DCOT instance by setting all node weights to 1, letting $\mathcal{D}^+(v) = \{1, \ldots, n-1\}$, and setting $t = 2k' \leq 2k$. In next section we show how to solve DCOT.

dcot$(G, w, \mathcal{D}^+, r, t)$

(1) If $t \le a$, solve the problem by brute force and return the obtained solution. If $M = 0$, return $(\{r\}, \{\})$.
(2) Compute a (n, t)-universal set $\mathcal{F}$ with the algorithm in [38].
(3) For any subset of nodes $S \subseteq V(G)$, with $r \in S$ and $|S| \le \log t + 1$, for any two out-degree assignments $d_L^+ : S \to \{0, \ldots, n-1\}$ and $d_R^+ : S \to \{0, \ldots, n-1\}$, for any two disjoint subsets $E_L, E_R \subseteq S \times S$ such that $E_L \cup E_R$ is an r-out-branching of $(S, S \times S)$, for any $f \in \mathcal{F}$:
  (3.a) Consider the partition (V_L, V_R) of $V - S$ induced by f.
  (3.b) Construct a graph G_L from G, by removing nodes V_R, adding edges E_R, and splitting those edges. Let D_R and F_R be the new nodes and edges, respectively, created by the splittings.
  (3.c) Define a node weight function w_L on G_L, with $w_L(v) = M' := M \cdot (n+1)$ for $v \in D_R \cup S$, and $w_L(v) = w(v)$ otherwise.
  (3.d) Define degree constraints $\mathcal{D}_L^+$ on G_L, with $\mathcal{D}_L^+(v) = \{d_L^+(v)\}$ for $v \in S$, $\mathcal{D}_L^+(v) = \{2\}$ for $v \in D_R$, and $\mathcal{D}_L^+(v) = \mathcal{D}^+(v)$ otherwise.
  (3.e) Compute $SOL'_L := \text{dcot}(G_L, w_L, \mathcal{D}_L^+, r, t_L)$, $t_L = (t + |S|)/2 + |D_R|$, and set $SOL_L := SOL'_L - D_R$.
  (3.f) Construct SOL_R symmetrically. Let $SOL := SOL_L \cup SOL_R$.
(4) Among the subgraphs SOL of G computed above, return a feasible solution of largest weight.

Fig. 3 Algorithm for DCOT. Here a is a sufficiently large constant, M denotes the largest node weight, and n the number of nodes

4.2 An Algorithm for DCOT

Our algorithm for DCOT is described in Fig. 3. If t is sufficiently small, the problem is solved by brute force (in polynomial time).

Otherwise the algorithm computes, using the algorithm in [38], an (n, t)-universal set $\mathcal{F}$. Then it generates a proper set of pairs of DCOT subproblems $(G_L, w_L, \mathcal{D}_L^+, r, t_L)$ and $(G_R, w_R, \mathcal{D}_R^+, r, t_R)$, and solved them recursively. The reasons behind the choice of the pairs will be clearer from the correctness analysis. One obtains a bipartition (V_L, V_R) of $V - S$ from $f \in F$ by placing $v \in V - S$ in set V_L if $f(v) = 0$, and in set V_R otherwise. Like for DCST, we set the weight of some nodes to a large value, and restrict the associated desirable degrees to a unique degree: this will force the solution to set the degree of those nodes accordingly.

Lemma 10 (Correctness) *Algorithm* dcot *returns a feasible solution of maximum weight.*

Proof If the algorithm returns a solution, it is feasible. Let us show by induction on t that the algorithm returns a feasible solution of maximum weight. The claim is trivially true if the algorithm halts at Step (1).

Otherwise, consider the following choice for the tuple $(S, d_L^+, d_R^+, E_L, E_R, f)$ (see also Fig. 2.b), with the corresponding digraphs SOL, SOL_L etc. Let OPT be the optimum solution. We let S be a minimum-cardinality perfectly balanced separator of OPT, with $r \in S$. The Sharp Separation Lemma guarantees the existence of such S, with $|S| \le \log t + 1$ (the $+1$ coming from r). Let W_L and W_R be the partition of OPT induced by S. We choose a function f such that $W_L \subseteq V_L$ and $W_R \subseteq V_R$. Note that, since $|W_R \cup W_L| \le t$, there must be an $f \in \mathcal{F}$ which satisfies this property. Let $OPT_L := OPT[W_L \cup S]$ and $OPT_R := OPT[W_R \cup S] - E(OPT[S])$. Observe that OPT_L and OPT_R bipartition the edges of OPT.

We next define E_R and d_L^+, the definition of E_L and d_R^+ being symmetric. Let us iteratively contract the edges of OPT_R which contain at least one node outside S (the new node inherits the label of the endpoint in S, if any). The resulting set of edges in $S \times S$ defines E_R. Let us remark that $E_L \cup E_R$ defines an r-out-branching on node set S, as required. For any $s \in S$, we set $d_L^+(s) := d_{OPT_L}^+(s) + d_{E_R}^+(s)$ (i.e., the out-degree of s in $OPT_L \cup E_R$, since OPT_L and E_R are edge-disjoint).

Let us first show that SOL is a feasible solution. By construction, SOL contains at most $(t - |S|)/2 + (t - |S|)/2 + |S| = t$ nodes. In order to show that SOL is an r-out-tree, it is sufficient to show that, for any $s \in S - \{r\}$, there is exactly one simple path from r to s. This can be proved exactly in the same manner as in the proof of Lemma 5, where pair (r, s) plays the role of pair $\{s', s''\}$, and directed paths replace undirected paths.

Observe that $OPT'_L := OPT_L \cup F_R$ is a feasible solution for the subproblem $(G_L, w_L, \mathcal{D}_L^+, r, t_L)$. In fact, it is an r-out-tree of G_L. Furthermore, it contains at most $(t - |S|)/2 + |S|$ nodes from OPT_L, and $|D_R|$ extra nodes from F_R. By the same approach as in Lemma 5, it is not hard to derive $w(\text{hit}(SOL_L) \cap V_L) \geq w(\text{hit}(OPT_L) \cap V_L)$. Symmetrically, $w(\text{hit}(SOL_R) \cap V_R) \geq w(\text{hit}(OPT_R) \cap V_R)$. Also in this case, each $s \in S$ has exactly the same degree in OPT_L and SOL_L (resp., OPT_R and SOL_R), which implies $H := \text{hit}(OPT) \cap S = \text{hit}(SOL) \cap S$. One can derive $w(\text{hit}(SOL)) \geq w(\text{hit}(OPT))$ by the same chain of inequalities as in Lemma 5. □

Lemma 11 (Running time) *The running time of* dcot *is* $O(4^t n^{O(\log^2 t)})$, *and its space complexity is* $O(2^t t^{O(\log t)} n \log n)$.

Proof Assume $n \geq 2$. We prove by induction on t that the running time $T(n, t)$ of the algorithm satisfies

$$T(n, t) \leq 4^t n^{b \log^2 t},$$

for a proper constant b. The claim is trivially true when the algorithm halts at Step (1).

Next assume $n \geq t \gg 1$, and consider an instance where the algorithm branches. The number of possible choices for the tuple $(S, d_L^+, d_R^+, E_L, E_R, f)$ is at most:

$$2\binom{n}{\log t + 1} \cdot n^{\log t + 1} \cdot n^{\log t + 1} \cdot \binom{(\log t + 1)^2}{\log t} 2^{\log t} \cdot 2^t t^{O(\log t)} \log n.$$

This is at most $2^t n^{c \log t}$ for a proper constant c, and it dominates the running time to construct the universal set. Observe that $t_L, t_R \leq (t + \log t + 1)/2 + \log t \leq t/2 + 2 \log t \leq 2t/3$. The overall running time therefore satisfies, for a constant b large enough,

$$\begin{aligned} T(n, t) &\leq 2^t n^{c \log t} \cdot (T(n, t_L) + T(n, t_R)) \leq 2^t n^{c \log t} \cdot 2 \cdot 4^{t/2 + 2 \log t} n^{b \log^2(2t/3)} \\ &\leq 4^t n^{c \log t + 1 + 2 \log t + b(\log t - \log(3/2))^2} \leq 4^t n^{b \log^2 t}. \end{aligned}$$

The space complexity is dominated by the space needed to store the universal sets in a chain of $O(\log t)$ recursive calls, which is at most

$$\sum_{i\geq 1} 2^{t/2^i} \cdot t^{O(\log t)} n \log n = O(2^t t^{O(\log t)} n \log n).$$

□

Theorem 12 *There is a deterministic algorithm which solves* k-INTERNAL OUT-BRANCHING *in* $O(16^{k+o(k)} + n^{O(1)})$ *time and* $O(4^{k+o(k)} + n^{O(1)})$ *space.*

Proof Consider the algorithm which first applies the reductions from Sect. 4.1, and then applies dcot. The reductions take polynomial time and space. The resulting DCOT instance contains $\overline{n} = O(k^2)$ nodes and has $t \leq 2k$. Hence the running time is $O(n^{O(1)} + 4^{2k} k^{O(\log^2 k)}) = O(16^{k+o(k)} + n^{O(1)})$, and the space complexity is $O(n^{O(1)} + 2^{2k} k^{O(\log k)}) = O(4^{k+o(k)} + n^{O(1)})$. □

4.3 Saving Space via Randomization

The space complexity can be made polynomial, without increasing the running time, by means of randomization. The resulting algorithm however might fail to find a feasible solution. The idea is replacing the universal set with a sufficiently large number N of random partitions of the node set. One can show that $N = \Theta(2^t)$ is sufficient to achieve a constant probability of success of the overall algorithm. We leave the details to the interested reader.

5 Conclusions

In this paper we proved a new, simple separation theorem for graphs of bounded treewidth, which turns out to be a useful tool in the design of divide-and-conquer algorithms, both exact (exponential) and parameterized. We demonstrated the applicability of our theorem by giving an algorithm for k-INTERNAL OUT-BRANCHING running in $O(16^{k+o(k)} + n^{O(1)})$ time and an algorithm for DEGREE CONSTRAINED SPANNING TREE running in time $O(2^{n+o(n)})$. It would be interesting to find further applications of our separation result in the fields of exact and parameterized algorithms.

References

1. Alon, N., Seymour, P., Thomas, R.: A separator theorem for non-planar graphs. J. Am. Math. Soc. **3**, 801–808 (1990)
2. Beigel, R., Eppstein, D.: 3-coloring in time $O(1.3289^n)$. J. Algorithms **54**(2), 168–204 (2005)
3. Björklund, A.: Determinant sums for undirected Hamiltonicity. In: IEEE Symposium on Foundations of Computer Science (FOCS), pp. 173–182 (2010)
4. Björklund, A., Husfeldt, T., Koivisto, M.: Set partitioning via inclusion-exclusion. SIAM J. Comput. **39**(2), 546–563 (2009)
5. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Fourier meets Möbius: Fast subset convolution. In: ACM Symposium on Theory of Computing (STOC), pp. 67–74 (2007)

6. Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Theor. Comput. Sci. **209**(1–2), 1–45 (1998)
7. Bourgeois, N., Escoffier, B., Paschos, V.T., van Rooij, J.M.M.: A bottom-up method and fast algorithms for max independent set. In: Scandinavian Symposium and Workshops on Algorithm Theory (SWAT), pp. 62–73 (2010)
8. Chen, J., Lu, S., Sze, S.-H., Zhang, F.: Improved algorithms for path, matching, and packing problems. In: ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 298–307 (2007)
9. Christofides, N.: An algorithm for the chromatic number of a graph. Comput. J. **14**(1), 38–39 (1971)
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. Diestel, R.: Graph Theory. Springer, Berlin (2010)
12. Fernau, H., Gaspers, S., Raible, D.: Exact and parameterized algorithms for max internal spanning tree. In: International Workshop on Graph-Theoretic Concepts in Computer Science (WG), pp. 100–111 (2009)
13. Fomin, F.V., Kratsch, D.: Exact Exponential Algorithms. Springer, Berlin (2010)
14. Fomin, F.V., Grandoni, F., Kratsch, D.: Faster Steiner tree computation in polynomial-space. In: European Symposium on Algorithms (ESA), pp. 430–441 (2008)
15. Fomin, F.V., Grandoni, F., Kratsch, D.: Solving connected dominating set faster than 2^n. Algorithmica **52**(2), 153–166 (2008)
16. Fomin, F.V., Grandoni, F., Kratsch, D.: A measure & conquer approach for the analysis of exact algorithms. Journal of the ACM **56**(5) (2009)
17. Fomin, F.V., Grandoni, F., Pyatkin, A.V., Stepanov, A.A.: Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. ACM Transactions on Algorithms **5**(1) (2008)
18. Fomin, F.V., Gaspers, S., Pyatkin, A.V., Razgon, I.: On the minimum feedback vertex set problem: Exact and enumeration algorithms. Algorithmica **52**(2), 293–307 (2008)
19. Fomin, F.V., Gaspers, S., Saurabh, S., Thomassé, S.: A linear vertex kernel for maximum internal spanning tree. In: International Symposium on Algorithms and Computation (ISAAC), pp. 275–282 (2009)
20. Fomin, F.V., Lokshtanov, D., Grandoni, F., Saurabh, S.: Sharp separation and applications to exact and parameterized algorithms. In: Latin American Symposium on Theoretical Informatics (LATIN), pp. 72–83 (2010)
21. Fuchs, B., Kern, W., Mölle, D., Richter, S., Rossmanith, P., Wang, X.: Dynamic programming for minimum steiner trees. Theory Comput. Syst. **41**(3), 493–500 (2007)
22. Fürer, M., Raghavachari, B.: Approximating the minimum-degree Steiner tree to within one of optimal. J. Algorithms **17**(3), 409–423 (1994)
23. Gaspers, S., Saurabh, S., Stepanov, A.A.: A moderately exponential time algorithm for full degree spanning tree. In: International Conference on Theory and Applications of Models of Computation (TAMC), pp. 479–489 (2008)
24. Goemans, M.X.: Minimum bounded degree spanning trees. In: IEEE Symposium on Foundations of Computer Science (FOCS), pp. 273–282 (2006)
25. Grandoni, F.: A note on the complexity of minimum dominating set. J. Discrete Algorithms **4**(2), 209–214 (2006)
26. Gutin, G., Razgon, I., Kim, E.J.: Minimum leaf out-branching and related problems. Theor. Comput. Sci. **410**(45), 4571–4579 (2009)
27. Held, M., Karp, R.M.: A dynamic programming approach to sequencing problems. J. SIAM **10**, 196–210 (1962)
28. Karp, R.M.: Dynamic programming meets the principle of inclusion and exclusion. Oper. Res. Lett. **1**, 49–51 (1982)
29. Khuller, S., Bhatia, R., Pless, R.: On local search and placement of meters in networks. SIAM J. Comput. **32**(2), 470–487 (2003)
30. Kneis, J., Mölle, D., Richter, S., Rossmanith, P.: Divide-and-color. In: International Workshop on Graph-Theoretic Concepts in Computer Science (WG), pp. 58–67 (2006)
31. Kohn, S., Gottlieb, A., Kohn, M.: A generating function approach to the traveling salesman problem. In: ACM Annual Conference, pp. 294–300 (1977)
32. Kullmann, O.: New methods for 3-SAT decision and worst-case analysis. Theor. Comput. Sci. **223**(1–2), 1–72 (1999)

33. Lawler, E.L.: A note on the complexity of the chromatic number problem. Inf. Process. Lett. **5**(3), 66–67 (1976)
34. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM J. Appl. Math. **36**, 177–189 (1979)
35. Lipton, R.J., Tarjan, R.E.: Applications of a planar separator theorem. SIAM J. Comput. **9**, 615–627 (1980)
36. Lokshtanov, D., Nederlof, J.: Saving space by algebraization. In: ACM Symposium on Theory of Computing (STOC), pp. 321–330 (2010)
37. Monien, B., Speckenmeyer, E.: Solving satisfiability in less than 2^n steps. Discrete Appl. Math. **10**(3), 287–295 (1985)
38. Naor, M., Schulman, L.J., Srinivasan, A.: Splitters and near-optimal derandomization. In: IEEE Symposium on Foundations of Computer Science (FOCS), pp. 182–191 (1995)
39. Nederlof, J.: Fast polynomial-space algorithms using Möbius inversion: Improving on Steiner tree and related problems. In: International Colloquium on Automata, Languages and Programming (ICALP), pp. 713–725 (2009)
40. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, London (2006)
41. Prieto, E., Sloper, C.: Reducing to independent set structure—the case of k-internal spanning tree. Nord. J. Comput. **12**(3), 308–318 (2005)
42. Razgon, I.: Exact computation of maximum induced forest. In: Scandinavian Symposium and Workshops on Algorithm Theory (SWAT), pp. 160–171 (2006)
43. Reed, B.A., Smith, K., Vetta, A.: Finding odd cycle transversals. Oper. Res. Lett. **32**(4), 299–301 (2004)
44. Robson, J.M.: Algorithms for maximum independent sets. J. Algorithms **7**(3), 425–440 (1986)
45. Singh, M., Lau, L.C.: Approximating minimum bounded degree spanning trees to within one of optimal. In: ACM Symposium on Theory of Computing (STOC), pp. 661–670 (2007)
46. Tarjan, R.E., Trojanowski, A.E.: Finding a maximum independent set. SIAM J. Comput. **6**(3), 537–546 (1977)
47. van Rooij, J.M.M., Bodlaender, H.L.: Design by measure and conquer, a faster exact algorithm for dominating set. In: Symposium on Theoretical Aspects of Computer Science (STACS), pp. 657–668 (2008)
48. Williams, R.: Finding paths of length k in $O^*(2^k)$ time. Inf. Process. Lett. **109**(6), 315–318 (2009)