Sharp Separation and Applications to Exact and Parameterized Algorithms

Fedor V. Fomin[1] Daniel Lokshtanov[1] Fabrizio Grandoni[2] Saket Saurabh[3]

[1] Department of Informatics, University of Bergen, N-5020 Bergen, Norway.
{fedor.fomin|daniello}@ii.uib.no
[2] Dipartimento di Informatica, Sistemi e Produzione, Università di Roma Tor Vergata,
via del Politecnico 1, 00133, Roma, Italy.
grandoni@disp.uniroma2.it
[3] The Institute of Mathematical Sciences, Chennai 600113, India.
saket@imsc.res.in

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 parameterized algorithm with running time $O(16^{k+o(k)} + n^{O(1)})$ for the Maximum Internal Subtree problem in directed graphs. This is a significant improvement over the best previously known parameterized algorithm for the problem by [Cohen et al.'09], running in time $O(49.4^k + n^{O(1)})$.
The second application is a $O(2^{n+o(n)})$ time and space 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, Maximum Internal Spanning Tree and their edge weighted variants.

1 Introduction

The aim of *parameterized* and *exact* algorithms is solving 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. Exact and parameterized algorithms have an old history [14, 18], 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, Möbius Transformation, Subset Convolution, Measure & Conquer and Iterative Compression to name a few [2, 3, 9, 17, 24].

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 solutions obtained. 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 $\alpha\, n$ such that no vertex of V_L is adjacent to any vertex of V_R. For example, the classical result of Lipton and Tarjan 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 $O(2^{O(\sqrt{n})})$ [19].

1.1 Our Results

In this paper (see Section 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. Given a function $w : X \to \mathbb{R}$, we define $w(Y) = \sum_{y \in Y} w(y)$ for any $Y \subseteq X$.

Theorem 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 $|S| \leq t\, p$, $w(V_L) \leq x + \frac{w(V)}{2^{p+1}}$, $w(V_R) \leq w(V) - x + \frac{w(V)}{2^{p+1}}$, 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 w is used to model a subset $W \subseteq V$ of vertices that we wish to separate. Theorem 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 set $p \geq \log w(V)$, which makes the additive term $w(V)/2^{p+1}$ disappear.

Our Sharp Separation Theorem 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.

k-Internal Spanning Tree Our first result is a parameterized algorithm for the following problem.

> k-Internal Out-Branching: Given a digraph $D = (N, A)$ and a positive integer k, check whether there exists an out-branching with at least k internal vertices.

The *undirected* counterpart to this problem, k-Internal Subtree was first studied by Prieto and Sloper [22], 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. [10] 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. [13] obtained an algorithm of running time $2^{O(k \log k)} n^{O(1)}$ for and gave a kernel of size of $O(k^2)$. A faster algorithm, running in time $49.4^k n^{O(1)}$ was subsequently improved by Cohen et al. [6]. In this paper we use the Sharp Separation Theorem to obtain an algorithm with running time $O(16^{k+o(k)} + n^{O(1)})$.

Theorem 2. *There is a one-sided-error Monte-Carlo algorithm for k-*Internal Out-Branching*. The algorithm runs in polynomial-space and in time $O(16^{k+o(k)} + n^{O(1)})$, where n is the size of the input digraph D. When an out-branching with at least k internal nodes exists the algorithm fails to find one with probability at most $1/4$. This algorithm can be derandomized at the cost of an exponential $O(4^k k^{O(\log k)})$ space complexity.*

Degree constrained spanning tree. The second application of the Sharp Separation Theorem is an algorithm for Degree Constrained Spanning Tree defined below. For a given graph $G = (V, E)$, let $d_G(v)$ denote the degree of $v \in V$ in G.

> 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. By carefully choosing the functions $\mathcal{D}(v)$ one can code many other problems as well, such as Full Degree Spanning Tree [16], Bounded Degree Spanning Tree [12] or Maximum Internal Spanning Tree [8]

For most special cases of DCST, no algorithm with running time $O(2^n n^{O(1)})$ was known until recently, and for Maximum Internal Spanning Tree Fernau et al. [8] give a $O(3^n n^{O(1)})$ time algorithm, leaving the existence of a $O(2^n n^{O(1)})$ time algorithm open.

This year Nederlof [21] was able to give Inclusion-Exclusion based algorithm running in time $O(2^n n^{O(1)})$ for DCST. We use the Sharp Separation Theorem to give an alternate algorithm for the DCST problem, in particular we prove the following result.

Theorem 3. [$\star$] [4] *The* Degree Constrained Spanning Tree *problem can be solved in time and space $O(2^{n+o(n)})$, where n is the number of nodes in the graph.*

[4] Proof of results labelled by $\star$ have been wholly or partially omitted due to space constraints

Our algorithm differs from the work of Nederlof in the following ways. On one hand, his algorithm takes polynomial space and works in $2^n n^{O(1)}$ time. On the other hand, our approach is more robust. In particular our algorithm can be easily extended to find subgraphs of constant treewidth instead of trees, and also works for edge weighted variants of DEGREE CONSTRAINED SPANNING TREE.

1.2 Preliminaries

For basic graph terminology we refer the reader, e.g., to [7]. We just recall the definition of treewidth, and also the less standard digraph notions needed in this paper.

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

1. $\bigcup_{i \in W} X_i = V$,
2. for each edge $vw \in E$, there is an $i \in W$ such that $v, w \in X_i$, and
3. 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. By a classical result of Arnborg, Corneil and Proskurowski [1], a tree-decomposition of G of width t, if any, can be computed in $O(n^{t+2})$ time. When this running time is dominated by other steps of the algorithm considered, we will just consider this decomposition as given. An *r-out-tree* in a digraph $D = (N, A)$ is a subtree T of D rooted at r, such that all arcs of T are oriented away from r. If T contains all vertices of D, T is said to be an *r-out-branching.* For a vertex set R, an *R-out-forest* is a collection of $|R|$ vertex-disjoint r-out-trees, one out-tree for each $r \in R$.

2 Sharp Separation in Graphs of Bounded Treewidth

In this section we prove our Sharp Separation Theorem, 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 1 ([4]). *Given a n-vertex graph $G = (V, E)$ of treewidth t and $w : V \to \mathbb{R}^+ \cup \{0\}$. There is a set T of vertices of size at most t 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.*

Proof. (Theorem 1) We construct V_L, V_R and S iteratively, starting from empty sets, as follows. By Lemma 1 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 $G[C_2]$. W.l.o.g. 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.

Each time the weight of C halves, so at the end of the process $w(C) \leq w(V)/2^{p+1}$. 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 t, we eventually obtain $|S| \leq p\,t$. The running time claim follows immediately from Lemma 1. This concludes the proof. □

3 *k*-Internal Out-Branching

In this section we use Theorem 1 to 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 Theorem with the *divide-and-color* paradigm in [5, 15] and a polynomial-sized kernel for the problem [13]. First we present a (polynomial-space) one-sided-error Monte-Carlo algorithm for k-Internal Out-Branching with the claimed running time. We then derandomize the algorithm at the cost of an exponential space complexity.

3.1 A Monte-Carlo Algorithm

The first step of our algorithm is to apply the *kernelization algorithm* of Gutin et al. [13]. Given an instance (D, k) of k-Internal Out-Branching the algorithm of Gutin et al. produces a new instance (D', k') with $|D'| = O(k^2)$ and $k' \leq k$ such that D' has an out-branching with at least k' internal vertices if and only if D has an out-branching with at least k internal vertices. After this step we can assume that the number n of vertices in the input digraph D is at most $O(k^2)$.

Now, the algorithm guesses the root r of the out-branching, and verifies that there indeed is some out-branching of D rooted at r. This guessing step, together with the following observation, allows us to search for out-trees rooted at r instead of out-branchings of D.

Lemma 2 ([6]). *Let D be a digraph and r be a node of D such that there is an r-out-branching of D. 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.*

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 nodes into leaves. We formalize this as an observation.

Lemma 3 ([6]). *Let D be a digraph and r be a node of D. 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.*

With the described preliminary steps, we have arrived at the following problem, which we call Rooted Directed k-Internal Out-Tree (k-RDIOT). Input is a digraph D, node r and integer k. The digraph D has $n = O(k^2)$ nodes and the objective is to decide whether there is an r-out-tree with at least k internal nodes and at most $2k$ nodes in total.

Our algorithm splits the original problem into two smaller sub-problems by means of a proper separator, guesses the "shape" of the intersection of the out-branching with each side of the separator and solves each subproblem recursively. There are two aspects of sub-problems which do not show up in the original problem. First of all, the solution to a subproblem is not necessarily an out-tree: it is an out-forest in general. Still, the union of such forests must induce an r-out-tree. In order to take this fact into account, we introduce the notion of signatures.

Definition 1. *Let $T = (N_T, A_T)$ be an R-out-forest, and $Z \subseteq N_T$ be a set of nodes such that $R \subseteq Z$. The* signature *$\zeta_Z(T)$ of T with respect to Z is the R-out-forest $C = (Z, Q)$ where there is an arc from a vertex $u \in R$ to a vertex $v \in Z \setminus R$ if and only if there is a path from u to v in T. All vertices of $Z \setminus R$ are leaves of C.*

Notice that the signature of an out-forest is always a set of stars and singletons. In our recursive steps we will guess the signature of the out-forest we are looking for with respect to Z, where the set Z includes r and all the separators guessed from the original problem down to the current subproblem.

Second, in order to obtain two independent sub-problems, we need to make sure that separator nodes that are internal on both sides of the separator only get counted once. To achieve this we guess a subset Y of the separator nodes, and do not count the internal nodes of the out-forest in Y. Altogether, a subproblem can be defined as follows.

> Directed Rooted Out-Forest (DROF). Input is a tuple (D, R, C, Y, k^*, t) where $D = (N, A)$ is a digraph, $C = (Z, Q)$ is an R-out-forest with node set Z for $R \subseteq Z \subseteq N$, $Y \subseteq Z$ is a node set and k^* and t are integers. The objective is to find an R-out-forest T in D with at least k^* internal nodes outside Y and at most t nodes outside Z such that T contains Z and $\zeta_Z(T) = C$.

The input instance (D, k) of k-RDIOT is equivalent to an DROF instance $(D, R, C, Y, k, 2k)$, where $t = 2k$, C is the single node r and $Y = \emptyset$. Our algorithm for k-RDIOT initially constructs a DROF instance equivalent to the input

k-RDIOT instance as described above. That k-RDIOT instance is solved recursively in the following way. Consider a given subproblem (D, R, C, Y, k^*, t). If $t \leq \log k$, that is the number of vertices outside Z in the out-forest sought for is small enough, we solve the problem by brute force. In particular, we enumerate all the possible R-out-forests in D on at most $|Z| + t$ nodes and check whether they satisfy the conditions of DROF.

Suppose now $t > \log k$, and that (D, R, C, Y, k^*, t) is a "yes"-instance. Then there is an R-out-forest $T = (N^T, A^T)$ that satisfies the conditions of DROF. By the Sharp Separation Theorem there is a partitioning of N^T into (N_L^T, S, N_R^T) such that $|S| = \log k$, $|N_L^T \setminus Z| \leq t/2$, $|N_R^T \setminus Z| \leq t/2$ and there are no arcs between N_L^T and N_R^T in T. Define $Z' = Z \cup S$ and $A_{Z'}$ to be the arcs of $T[Z']$. The algorithm guesses the separator $S \subseteq N$ and for each of the $\binom{O(k^2)}{\log k}$ guesses for the separator it generates a random family of $3 \cdot 2^t \cdot |Z'|^{O(|Z'|)}$ pairs of sub-problems, that is instances of DROF $\mathcal{P}_L$ and $\mathcal{P}_R$, which are solved recursively.

If for some pair $\mathcal{P}_L$ and $\mathcal{P}_R$, the algorithm returns that both $\mathcal{P}_L$ and $\mathcal{P}_R$ are "yes" instances, then the algorithm returns that (D, C, Y, k^*, t) is a "yes"-instance as well. If the algorithm loops through all guesses of S and all the $3 \cdot 2^t \cdot |Z'|^{O(|Z'|)}$ pairs and for each pair the algorithm returns that at least one sub-problem is a "no"-instance, the algorithm returns that (D, C, Y, k^*, t) is a "no"-instance. To conclude the description of the algorithm we need to describe how the pairs $(\mathcal{P}_L, \mathcal{P}_R)$ are generated.

Before describing how the pairs are generated, define the out-forests $T_L = T[Z' \cup N_L^T]$ and $T_R = T[Z' \cup N_R^T] \setminus A_{Z'}$. Also, let Y_R be Y plus all the internal nodes of T_L in Z' and $Y_L = (Z' \setminus Y_R) \cup Y$. Now, t_L and t_R are the number of nodes outside Z' in T_L and T_R respectively. Finally k_L^* and k_R^* are the number of internal nodes in T_L outside Y_L and the number of internal nodes in T_R outside Y_R respectively.

We next describe how a random pair $(\mathcal{P}_L, \mathcal{P}_R)$ is generated. The algorithm generates the pairs in $3 \cdot 2^t$ groups, each group with $|Z'|^{O(|Z'|)}$ pairs. For each group the algorithm partitions the node set $N \setminus Z'$ into two parts (N_L, N_R) uniformly at random. For each partitioning, the algorithm guesses $C_L = \zeta_{Z'}(T_L)$, $C_R = \zeta_{Z'}(T_R)$, Y_L, Y_R, k_L^*, k_R^*, t_L and t_R. Each set of guesses makes one pair $(\mathcal{P}_L, \mathcal{P}_R)$ of instances, where $\mathcal{P}_L = (D[N_L \cup Z'], R_L, C_L, Y_L, k_L^*, t_L)$ and $\mathcal{P}_R = (D[N_R \cup Z'], R_R, C_R, Y_R, k_R^*, t_R)$. It is easy to see that the number of possible guesses is at most $|Z'|^{O(|Z'|)}$.

The algorithm makes the guesses in a special way, making sure that if both $\mathcal{P}_L$ and $\mathcal{P}_R$ are "yes"-instances then (D, C, Y, k^*, t) is a "yes"-instance as well. In particular, it makes sure that the arc sets of C_L and C_R are disjoint, that $C_L \cup C_R$ is an out-forest and that $\zeta(C_L \cup C_R) = C$. Also, the algorithm makes sure that $Y_L \cup Y_R = Z'$ and that $Y \subseteq Y_L$ and $Y \subseteq Y_R$. Finally, it makes sure that $t_L^* + t_R^* - |Z'| = t^*$ and that $k_L^* + k_R^* = k^*$. This concludes the description of the algorithm.

Lemma 4. *There is a one-sided-error Monte-Carlo algorithm for k-*Internal Out-Branching *running in time $O(16^{k+o(k)} + n^{O(1)})$. When the instance is a* *"yes"-instance, the algorithm incorrectly returns "no" with probability at most* 1/4.

Proof. Consider the algorithm above. It is enough to prove correctness and analyze the running time for the part of the algorithm that solves DROF. We first prove that when the algorithm answers yes, the answer is correct. We prove this by induction on t. If $t < \log k$ then the algorithm resolves the problem in a brute force manner and hence correctness follows. Suppose now that $t \geq \log k$. Since the algorithm returned yes it made a guess for S, a random partitioning of $N \setminus Z'$ (where $Z' = Z \cup S$) and guessed a pair $\mathcal{P}_L = (D[N_L \cup Z'], R_L, C_L, Y_L, k^*_L, t_L)$ and $\mathcal{P}_R = (D[N_R \cup Z'], R_R, C_R, Y_R, k^*_R, t_R)$ such that the algorithm returned that both $\mathcal{P}_L$ and $\mathcal{P}_R$ are "yes"-instances of DROF. By the induction hypothesis there are out-forests $T_L = (N^T_L, A^T_L)$ of $D[N_L]$ and $T_R = (N^T_R, A^T_R)$ of $D[N_R]$ with at least k^*_L and k^*_R inner nodes outside Y_L and Y_R respectively, such that $C_{Z'}(T_L) = C_L$ and $C_{Z'}(T_R) = C_R$. We prove that $T = T_L \cup T_R$ is an out-forest that satisfies the conditions of DROF.

Since the arc sets of C_L and C_R are disjoint, $C_L \cup C_R$ is an out-forest and $C_{Z'}(T_L) = C_L$ and $C_{Z'}(T_R) = C_R$, $T = T_L \cup T_R$ is an out-forest. Since $\zeta_Z(C_L \cup C_R) = C$ it follows that $\zeta_Z(T) = \zeta_Z(T_L \cup T_R) = C$. The number of nodes in T is $t_L + t_R - Z \leq t$ and since $Y \subseteq Y_L$, $Y \subseteq Y_R$ and $Y_L \cup Y_R = Z'$ the number of inner nodes of T avoiding Y is at most $k^*_L + k^*_R \geq k^*$. Hence the input instance is indeed a "yes"-instance.

Now, we prove that if a given subproblem (D, R, C, Y, k^*, t) is a "yes"-instance, then the probability that the algorithm returns "no" is $p_t \leq 1/4$. We prove this by induction on t, and if $t < \log k$ the algorithm solves the problem by brute force and correctness follows. If $t \geq \log k$, consider an out-forest T that satisfies the conditions of DROF. Consider the run of the algorithm where the separator S is guessed correctly.

Now, there are two possible reasons why the algorithm fails to answer "yes". Reason (a) is that the random partition (N_L, N_R) of $N \cup Z'$ could be done in the wrong way, that is $N^T_L \not\subseteq N_L$ or $N^T_R \not\subseteq N_R$. Reason (b) is that even though N_L and N_R are guessed correctly, in the iteration of the algorithm where the guesses for $C_L = \zeta_{Z'}(T_L)$, $C_R = \zeta_{Z'}(T_R)$, Y_L, Y_R, k^*_L, k^*_R, t_L and t_R are correct, the algorithm could fail to recognize either $\mathcal{P}_L$ or $\mathcal{P}_R$ as "yes" instances.

The probability of the first event is at most $1 - 2^{-t}$. Recall that $t_L, t_R \leq t/2$, since the algorithm uses a perfectly balanced separator to split $N^T \setminus Z'$. Hence, by the union bound, the probability of event (b) is at most $2^{-t}\, 2p_{t/2}$. Altogether p_t satisfies

$$p_t \leq \left(1 - 2^{-t} + 2^{-t+1} p_{t/2}\right)^{3 \cdot 2^t}.$$

Therefore, by the inductive hypothesis,

$$p_t \leq \left(1 - 2^{-t} + 2^{-t+1}/4\right)^{3 \cdot 2^t} = \left(\left(1 - 1/2^{t+1}\right)^{2^{t+1}}\right)^{1.5} \leq e^{-1.5} \leq \frac{1}{4}.$$

Consider now the running time of the algorithm. Observe that in the beginning $t = 2k$ and that t always drops by a factor of one half in the recursive steps. Furthermore the algorithm stops when t drops below $\log k$. Hence the recursion depth is at most $\log(2k)$. For each new level of the recursion the size of Z' increases by $\log 2k$. Hence $|Z'|$ never grows over $\log^2(2k)$. In the base case we try all possible subsets of A of size $|Z|' + t$. Since D has at most $O(k^2)$ vertices it has at most $O(k^4)$ arcs and hence in the base we need to try at most $O(\binom{k^4}{\log^2 2k}) = O(2^{o(k)})$ different possibilities, each of which can be checked in $O(k^{O(1)})$ time.

Consider now the recursive step. There are $\binom{O(k^2)}{\log 2k}$ choices for the separator. For each choice of the separator the number of random partitions tried is $3 \cdot 2^k$. For each random partition, $|Z'|^{O(|Z'|)} = O(2^{\log^3 k})$ pairs $(\mathcal{P}_L, \mathcal{P}_R)$ of instances are generated. Let $T(n,t)$ be the running time of the DROF algorithm on an instance where D has n nodes and the number of nodes in the tree searched for that are not in Z' is t. Then the following recurrence holds.

$$\begin{aligned} T(n,t) &\leq n^{O(\log^3 2k)} \cdot 3 \cdot 2^t \cdot (2T\,(n, t/2) + n^{O(1)}) \\ &\leq n^{O(t \log^3 2k)} 2^k \cdot T\,(n, t/2) \\ &= O((n^{O(t \log^3 2k)})^{\log k} \cdot 2^{\left(\sum_{i=0}^{\log t} \frac{t}{2^i}\right)}) = O(4^t \cdot n^{O(\log^4 k)}). \end{aligned}$$

Since we first run the kernelization algorithm from [13], the k-RDIOT instance we solve recursively has $O(k^2)$ nodes. Since $t = 2k$ in the instance of DROF we construct from this k-RDIOT instance, the total running time for the algorithm is bounded from above by $O(4^{2k} \cdot (k^2)^{O(\log^4 k)} + n^{O(1)}) = O(16^{k+o(k)} + n^{O(1)})$. □

Our algorithm for k-Internal Out-Branching can be derandomized using the method presented by Chen et al. [5], which is based on the construction of (n,k)-*universal sets* [20]. The main idea is to replace the random partitioning of the host graph H by a partitioning that uses universal sets. Lemmas 4 and 5 together imply Theorem 2.

Lemma 5. [$\star$] *There is a deterministic algorithm for k-*Internal Out-Branching *running in time $O(16^{k+o(k)} + n^{O(1)})$ and requiring $O(4^k k^{O(\log k)})$ space.*

4 Degree Constrained Spanning Tree

In this section we present our $O(2^{n+o(n)})$-time algorithm for the Degree Constrained Spanning Tree problem (DCSS). We recall that in this problem we are given an undirected graph $G = (V, E)$, and a list of *desirable* degrees $\mathcal{D}(v)$ for each vertex v. The aim is finding a spanning tree T of G which maximizes the number of *hits*, i.e. the number of vertices v with $d_T(v) \in \mathcal{D}(v)$.

Our algorithm is based on the divide-and-conquer approach, and has several similarities with the algorithm for k-Internal Spanning Tree. The main differences are that the random partitioning and kernelization parts are no longer required, and that the Sharp Separation theorem is used to divide the problem into *very unbalanced* subproblems. Consider a subproblem on the graph

$H = (V, E)$. In the divide step we guess a proper (logarithmic-size) separator S of the optimum solution, and the corresponding two sides V_L and V_R of the partition. Set S is chosen such that V_L is sufficiently small to make the guessing of S, V_L and V_R cheap enough. The existence of S is guaranteed by our Sharp Separation Theorem. The two sub-problems induced by $V_L \cup S$ and $V_R \cup S$ are then solved recursively.

Just as for the case of k-Internal Spanning Tree there are two aspects of sub-problems which do not show up in the original problem. First of all, the solution to a subproblem is not necessarily a spanning tree: it is a spanning forest in general. Still, the union of such forests must induce a tree. In order to take this fact into account, we introduce a *constraint forest* $C = (Z, Q)$, defined over a proper subset of nodes $Z \subseteq V$. The set Z includes all the separators guessed from the original problem down to the current subproblem. The components of C describe which pairs of nodes of Z must and must not be connected in the desired forest.

Second, in order to obtain two independent maximization sub-problems, we need to guess the degree of the separator nodes in the optimum solution, and force the solution to have that degree on those nodes. This is modeled via an auxiliary function $\mathcal{A} : V \to 2^{\{1,...,n\}}$. For $z \in Z$, $\mathcal{A}(z)$ is a singleton set containing the mentioned guessed degree, while $\mathcal{A}$ coincides with $\mathcal{D}$ on the remaining nodes. We remark that it might be that $\mathcal{A}(z) \nsubseteq \mathcal{D}(z)$ for some $z \in Z$, since not all the nodes of Z need to be hits in the optimum solution. Altogether, a subproblem $(H, C, \mathcal{A})$ can be defined as follows.

Degree-Constrained Cut & Connect (DCCC). Given a graph $H = (V, E)$, a forest $C = (Z, Q)$, $Z \subseteq V$, and a function $\mathcal{A} : V \to 2^{\{1,...,n\}}$, $|\mathcal{A}(z)| = 1$ for $z \in Z$. Find a spanning forest F of H maximizing the number of hits, i.e. $|\{v \in V : d_F(v) \in \mathcal{A}(v)\}|$, such that: (i) every connected component of F contains at least one vertex of Z; (ii) for any $u, v \in Z$, u and v are connected in C if and only if they are connected in F; (iii) $d_F(z) \in \mathcal{A}(z)$ for all $z \in Z$.

Observe that the original Degree Constrained Spanning Tree instance $(G, \mathcal{D})$ is equivalent to a Degree-Constrained Cut & Connect instance where $H = G$, $C = (\{z\}, \emptyset)$ for an arbitrary vertex z of G, $\mathcal{A}(z) = \{d_{OPT}(z)\}$ where $d_{OPT}(z)$ is the degree of z in an optimum solution OPT, and $\mathcal{A}(v) = \mathcal{D}(v)$ for any vertex $v \neq z$. We remark that we can guess $d_{OPT}(z)$ by trying all the possibilities.

We give a memoization based algorithm for DCST. Initially the algorithm encodes the input problem into a DCCC problem as described above. The latter problem is then solved recursively. The solution to each subproblem generated is stored in a memoization table, which is used to avoid to solve the same sub-problem twice.

Let us describe the recursive algorithm for DCCC. Consider a given sub-problem $\mathcal{P} = (H, C, \mathcal{A})$, with $H = (V, E)$ and $C = (Z, Q)$. If $|V| \leq n/\log^2 n$, the problem is solved in a brute force manner by enumerating all the spanning forests F of H. Otherwise, the algorithm splits the problem in two smaller independent sub-problems $\mathcal{P}_L = (H_L, C_L, \mathcal{A}_L)$ and $\mathcal{P}_R = (H_R, C_R, \mathcal{A}_R)$, which are solved recursively. The desired solution F is obtained by merging the two solutions obtained for the two sub-problems.

We next describe how $\mathcal{P}_L$ and $\mathcal{P}_R$ are obtained. Consider the optimum solution $OPT = OPT(H, C, \mathcal{A})$ to $(H, C, \mathcal{A})$. For $x = n/\log^2 n$, by the Sharp Separation Theorem there is a separator S of OPT, $|S| \leq t \log n = \log n$, which splits $V \setminus S$ in two subsets V_L and V_R, with $|V_L| \leq x$ and $|V_R| \leq |V| - x$. Let $Z' = S \cup Z$. The algorithm guesses S, V_L and V_R, and sets $H_L = H[V_L \cup Z']$ and $H_R = H[V_R \cup Z']$.

Consider the forest C' obtained from OPT by iteratively contracting the edges with one endpoint not in Z'. Note that, if we further contract C' on vertices $S \setminus Z$, we must obtain the forest C. Each edge of C' corresponds to a path in H whose vertices belong entirely either to $V_L \cup Z'$ or to $V_R \cup Z'$. (In order to simplify the algorithm, we assume that edges between adjacent nodes of Z' belong to the first class). Let Q_L and Q_R be the edges of the first and second type, respectively. The algorithm guesses C', Q_L and Q_R, and sets $C_L = (Z', Q_L)$ and $C_R = (Z', Q_R)$.

It remains to specify $\mathcal{A}_L$ and $\mathcal{A}_R$. Consider the two forests OPT_L and OPT_R, on vertex set $V_L \cup Z'$ and $V_R \cup Z'$, respectively, obtained from OPT by inserting every edge of OPT with both endpoints in $V_L \cup Z'$ in OPT_L, and all the remaining edges in OPT_R. Note that $d_{OPT}(z') = d_{OPT_L}(z') + d_{OPT_R}(z')$ for all $z' \in Z'$. The algorithm guesses $d_{OPT_L}(z')$ (resp., $d_{OPT_R}(z')$) for all $z' \in Z'$, and sets $\mathcal{A}_L(z') = \{d_{OPT_L}(z')\}$ (resp., $\mathcal{A}_R(z') = \{d_{OPT_R}(z')\}$). Moreover, it sets $\mathcal{A}_L(v) = \mathcal{A}(v)$ (resp, $\mathcal{A}_R(v) = \mathcal{A}(v)$) for all the remaining nodes v.

Summarizing the discussion above, the following recurrence holds, where the maximum, computed with respect to the number of hits, is taken over all the possible choices of $(H_L, C_L, \mathcal{A}_L)$ and $(H_R, C_R, \mathcal{A}_R)$ such that the pair of feasible solutions to the smaller instances can be combined to a feasible solution for the original instance $(H, C, \mathcal{A})$.

$$OPT(H, C, \mathcal{A}) = \arg\max\{OPT(H_L, C_L, \mathcal{A}_L) \cup OPT(H_R, C_R, \mathcal{A}_R)\}. \tag{1}$$

In particular, the maximum considers all the possible choices of the separator S and of the partition (V_L, V_R), of the forest C' and of the partition (Q_L, Q_R) of its edges, and of the degrees $d_{OPT_L}(z')$ and $d_{OPT_R}(z')$.

Due to correctness of Recurrence (1) the algorithm described above solves Degree Constrained Spanning Tree. What remains for a complete proof of Theorem 3 is a running time analysis, which has been omitted due to space restrictions.

Remark: The algorothm for Degree Constrained Spanning Tree can be applied to find spanning subgraphs of treewidth t satisfying degree constraints in time $O(2^{n+o(n)})$ for every fixed constant t. In addition to the degree constraints one could require the spanning subgraph to belong to a minor-closed graph family. Our approach is also easily generalizable to handle super-polynomial edge weights.

References

1. S. Arnborg, D. G. Corneil, and A. Proskurowski, Complexity of finding embeddings in a k-tree. SIAM J. Algebraic Discrete Methods, 8 (1987), 277–284.
2. A. Björklund, T. Husfeldt, Inclusion–Exclusion Algorithms for Counting Set Partitions. In FOCS (2006), 575–582.
3. A. Björklund, T. Husfeldt, P. Kaski and M. Koivisto, Fourier meets Möbius: Fast Subset Convolution. In STOC (2007), 67–74.
4. H. L. Bodlaender, A partial k-arboretum of graphs with bounded treewidth. Theor. Comp. Sc., 209 (1998), 1–45.
5. J. Chen, S. Lu, S.-H. Sze, and F. Zhang, Improved Algorithms for Path, Matching, and Packing Problems. In SODA (2007), 298-307.
6. N. Cohen, F. V. Fomin, G. Gutin, E. J. Kim, S. Saurabh and A. Yeo, Algorithm for Finding k-Vertex Out-trees and its Application to k-Internal Out-branching Problem. In COCOON (2009), 37–46
7. R. Diestel, Graph Theory. Springer (2005).
8. H. Fernau, D. Raible, S. Gaspers, A. A. Stepanov, Exact Exponential Time Algorithms for Max Internal Spanning Tree. In WG (2009). LNCS 5911, 100–111.
9. F. V. Fomin, F. Grandoni, D. Kratsch, A Measure & Conquer Approach for the Analysis of Exact Algorithms. To appear in J. ACM. Preliminary version in ICALP'05 and SODA'06.
10. F. V. Fomin, S. Gaspers, S. Saurabh and S. Thomassé. A Linear Vertex Kernel for Maximum Internal Spanning Tree. In ISAAC (2009), LNCS 5878, 275-282.
11. S. Gaspers, S. Saurabh, and A. A. Stepanov, A Moderately Exponential Time Algorithm for Full Degree Spanning Tree. In TAMC (2008), LNCS 4978, 479-489.
12. M. X. Goemans, Minimum bounded degree spanning trees. In FOCS (2006), 273–282.
13. G. Gutin, I. Razgon and E. J. Kim,Minimum Leaf Out-Branching Problems. In AAIM 2008, 235–246.
14. M. Held and R. M. Karp, A dynamic programming approach to sequencing problems. Journal of SIAM, 10 (1962), 196–210.
15. J. Kneis, D. Molle, S. Richter, and P. Rossmanith, Divide-and-color. In WG (2006), LNCS 4271, 58–67.
16. S. Khuller, R. Bhatia, and R. Pless, On local search and placement of meters in networks. SIAM J. Comput., 32 (2003), 470–487.
17. M. Koivisto, An $O(2^n)$ Algorithm for Graph Colouring and Other Partitioning Problems via Inclusion-Exclusion. In FOCS (2006), 583–590.
18. E. L. Lawler, A Note on the Complexity of the Chromatic Number Problem. Inform. Proc. Letters, 5 (3) (1976), 66–67.
19. ———, Applications of a planar separator theorem. SIAM J. Comput., 9 (1980), 615–627.
20. M. Naor, L. J. Schulman and A. Srinivasan, Splitters and Near-Optimal Derandomization. In FOCS (1995), 182–193.
21. J. Nederlof, Fast polynomial-space algorithms using Mobius inversion: Improving on Steiner Tree and related problems. In ICALP(1) (2009), 713–725.
22. E. Prieto and Christian Sloper, Reducing to Independent Set Structure – the Case of k-Internal Spanning Tree. Nord. J. Comput, 12 (3) (2005), 308–318
23. R. Niedermeier, Invitation to fixed-parameter algorithms, vol. 31 of Oxford Lecture Series in Mathematics and its Applications, Oxford University Press (2006).
24. B. Reed, A. Vetta and K. Smith, Finding Odd Cycle Transversals, Operations Research Letters, 32, 229–301 (2004).