Exact and Parameterized Algorithms for Max Internal Spanning Tree*

Daniel Binkele-Raible[1], Henning Fernau[1], Serge Gaspers[2], and Mathieu Liedloff[3]

[1] Univ. Trier, FB 4—Abteilung Informatik, D-54286 Trier, Germany
fernau@uni-trier.de, raible@informatik.uni-trier.de
[2] CMM – Univ. de Chile, 8370459 Santiago de Chile
sgaspers@dim.uchile.cl
[3] LIFO, Univ. d'Orléans, 45067 Orléans Cedex 2, France
liedloff@univ-orleans.fr

Abstract. We consider the $\mathcal{NP}$-hard problem of finding a spanning tree with a maximum number of internal vertices. This problem is a generalization of the famous Hamiltonian Path problem. Our dynamic-programming algorithms for general and degree-bounded graphs have running times of the form $\mathcal{O}^*(c^n)$ with $c \leq 2$. For graphs with bounded degree, $c < 2$. The main result, however, is a branching algorithm for graphs with maximum degree three. It only needs polynomial space and has a running time of $\mathcal{O}(1.8669^n)$ when analyzed with respect to the number of vertices. We also show that its running time is $2.1364^k n^{\mathcal{O}(1)}$ when the goal is to find a spanning tree with at least k internal vertices. Both running time bounds are obtained via a Measure & Conquer analysis, the latter one being a novel use of this kind of analysis for parameterized algorithms.

1 Introduction

Motivation. We investigate the following problem:

Max Internal Spanning Tree (MIST)
Given: A graph $G = (V, E)$ with n vertices and m edges.
Task: Find a spanning tree of G with a maximum number of internal vertices.

MIST is a generalization of the famous and well-studied Hamiltonian Path problem. Here, one is asked to find a path in a graph such that every vertex is visited exactly once. Clearly, such a path, if it exists, is also a spanning tree, namely one with a maximum number of internal vertices. Whereas the running time barrier of 2^n has not been broken for general graphs, Hamiltonian Path has faster algorithms for graphs of bounded degree. It is natural to ask if for the generalization, MIST, this can also be obtained.

A second issue is whether we can find an algorithm for MIST with a running time of the form $\mathcal{O}^*(c^n)$ at all. [1] The very naïve approach gives only an upper bound of $\mathcal{O}^*(2^m)$. A possible application is the following scenario. Suppose you have a set of cities which should be connected with water pipes. The cities and all possible connections between them can be represented by a graph G. It suffices to compute a spanning tree T for G. In T we may have high degree vertices that have to be implemented by branching pipes. These branching pipes cause turbulences and therefore pressure may drop. To minimize the number of branching pipes one can equivalently

* This work was partially supported by a PPP grant between DAAD (Germany) and NFR (Norway). A preliminary version of this paper appeared in the proceedings of WG 2009 [7].

[1] Throughout the paper, we write $f(n) = \mathcal{O}^*(g(n))$ if $f(n) \leq p(n) \cdot g(n)$ for some polynomial $p(n)$.

compute a spanning tree with the smallest number of leaves, leading to MIST. Vertices representing branching pipes should not be of arbitrarily high degree, motivating the investigation of MIST on degree-restricted graphs.[2]

Previous Work. It is well-known that the more restricted problem, Hamiltonian Path, can be solved in $\mathcal{O}(2^n n^2)$ steps and exponential space. This result has independently been obtained by R. Bellman [1], and M. Held and R. M. Karp [11]. The Traveling Salesman problem (TSP) is very closely related to Hamiltonian Path. Basically, the same algorithm solves this problem, but there has not been any improvement on the running time since 1962. The space requirements have, however, been improved and now there are $\mathcal{O}^*(2^n)$ algorithms needing only polynomial space: In 1977, S. Kohn et al. [15] gave an algorithm based on generating functions with a running time of $\mathcal{O}(2^n n^3)$ and space requirements of $\mathcal{O}(n^2)$ and in 1982 R. M. Karp [13] came up with an algorithm which improved storage requirements to $\mathcal{O}(n)$ and preserved this running time by an inclusion-exclusion approach.

D. Eppstein [6] studied TSP on cubic graphs. He achieved a running time of $\mathcal{O}(1.260^n)$ using polynomial space. K. Iwama and T. Nakashima [12] improved this to $\mathcal{O}(1.251^n)$. A. Björklund et al. [4] considered TSP with respect to degree-bounded graphs. Their algorithm is a variant of the classical 2^n-algorithm and the space requirements are therefore exponential. Nevertheless, they showed that for a graph with maximum degree d, there is a $\mathcal{O}^*((2-\epsilon_d)^n)$-algorithm. In particular for $d = 4$, there is an $\mathcal{O}(1.8557^n)$- and for $d = 5$ a $\mathcal{O}(1.9320^n)$-algorithm. Using an inclusion-exclusion approach, J. Nederlof [17] developed (independently and in parallel to the present paper and its conference version predecessor) an algorithm solving MIST on general graphs in time $\mathcal{O}^*(2^n)$ and polynomial-space. Based on a separation property, Fomin et al. [10] recently used a *divide & conquer* approach to solve a generalization of MIST in $O(2^{n+o(n)})$ time.

MIST was also studied with respect to parameterized complexity. The (standard) parameterized version of the problem is parameterized by k, and asks whether G has a spanning tree with at least k internal vertices. E. Prieto and C. Sloper [19] proved a $\mathcal{O}(k^3)$-vertex kernel for the problem showing $\mathcal{FPT}$-membership. The same authors [20] improved the kernel size to $\mathcal{O}(k^2)$ and F.V. Fomin et al. [9] to $3k$. Parameterized algorithms for MIST have been studied in [5, 9, 20]. E. Prieto and C. Sloper [20] gave the first FPT algorithm, with running time $2^{4k \log k} \cdot n^{\mathcal{O}(1)}$. This result was improved by N. Cohen et al. [5], who solve a more general directed version of the problem in time $49.4^k \cdot n^{\mathcal{O}(1)}$. The currently fastest algorithm for MIST has running time $8^k \cdot n^{\mathcal{O}(1)}$ [9] and the currently fastest algorithm for the directed version has running time $16^{k+o(k)} + n^{\mathcal{O}(1)}$ [10].

G. Salamon [22] studied the problem considering approximation. He achieved a $\frac{7}{4}$-approximation on graphs without pendant vertices. A $2(\Delta-3)$-approximation for the node-weighted version was a by-product. These results was further improved by M. Knauer and J. Spoerhase: In [14] they showed that a version of Salamon's algorithm leads to a $\frac{5}{3}$-approximation on general graphs and proved a ratio of $3 + \epsilon$ for the node-weighted version. Cubic and claw-free graphs were considered by G. Salamon and G. Wiener [21]. They introduced algorithms with approximation ratios $\frac{6}{5}$ and $\frac{3}{2}$, respectively. Further variants of our problem are discussed in a survey [23], where more pointers to the literature can be found.

[2] This motivation can be derived from documents like `http://www.adpf.ae/images/Page-A.pdf`: "Pressure drop or head loss, occurs in all piping systems because of elevation changes, turbulence caused by abrupt changes in direction, and friction within the pipe and fittings." In the literature of water management etc., these types of pressure losses are usually qualified as minor, but this is only meant in comparison with the major loss due to the lengths of the pipes. When the locations are fixed, these lengths cannot be influenced any longer, so that the optimization of minor losses becomes a crucial factor. Also confer to textbooks like [16].

Our Results. Two algorithms are presented:

(*a*) A dynamic-programming algorithm, combined with a fast subset convolution evaluation, solving MIST in time $\mathcal{O}^*(2^n)$. We extend this algorithm and show that for any degree-bounded graph a running time of $\mathcal{O}^*((2-\epsilon)^n)$ with $\epsilon > 0$ can be achieved.
(*b*) A branching algorithm solving the problem for maximum degree 3 graphs in time $\mathcal{O}(1.8669^n)$. The space requirements are only polynomial in this case. For that algorithm, we provide a graph family proving a lower bound on the running time as well.
(*c*) We also analyze the same branching algorithm from a parameterized point of view, achieving a running time of $2.1364^k n^{\mathcal{O}(1)}$ to find a spanning tree with at least k internal vertices (if the graph admits such a spanning tree). The latter analysis is novel in a sense that we use a potential function analysis —*Measure & Conquer* — in a way that, to our knowledge, is much less restrictive than any previous analysis for parameterized algorithms that were based on the potential function method.

Notions and Definitions. We consider only simple undirected graphs $G = (V, E)$. The *neighborhood* of a vertex $v \in V$ in G is $N_G(v) := \{u \mid \{u, v\} \in E\}$ and its *degree* is $d_G(v) := |N_G(v)|$. The *closed neighborhood* of v is $N_G[v] := N_G(v) \cup \{v\}$ and for a set $V' \subseteq V$ we let $N_G(V') := \left(\bigcup_{u \in V'} N_G(u)\right) \setminus V'$. We omit the subscripts of $N_G(\cdot)$, $d_G(\cdot)$, and $N_G[\cdot]$ when the graph is clear from the context. For a subset of edges $E' \subseteq E$ we also write $N_{E'}(\cdot)$, $d_{E'}(\cdot)$, and $N_{E'}[\cdot]$ instead of $N_{(V,E')}(\cdot)$, $d_{(V,E')}(\cdot)$, and $N_{(V,E')}[\cdot]$. A *subcubic graph* has maximum degree at most three. In a tree T, vertices of degree 1 are called *leaves* and vertices of degree at least 2 are called *internal* vertices. Let $internal(T)$ denote the set of internal vertices and $leaves(T)$ the set of leaves of T. An *i-vertex* u is a vertex with $d_T(u) = i$ with respect to some subtree T of G. The *tree-degree* of some $u \in V(T)$ is $d_T(u)$. We also speak of the *T-degree* $d_T(v)$ when we refer to a specific subtree. A *Hamiltonian path* is a sequence of pairwise distinct vertices $v_1, \ldots, v_n$ from V such that $\{v_i, v_{i+1}\} \in E$ for $1 \leq i \leq n-1$. A *triangle* in a graph is a subgraph of the form $(\{a, b, c\}, \{\{a, b\}, \{b, c\}, \{a, c\}\})$. A *spanning tree* of G is a subtree of G on $|V|$ vertices. We also introduce the notion of *constrained spanning tree*: given a graph $G = (V, E)$, a vertex v and a spanning tree T of G, we say that T is a (v^L)-constrained spanning tree (shortly, (v^L)-cst) if v is a leaf in T. The notion of (v^I)-constrained spanning tree, denoted (v^I)-cst, is defined similarly by requiring that v is an internal node in the spanning tree. A (v^L)-cst ((v^I)-cst) T of G with a minimum number of leaves (or a maximum number of internal vertices) is a spanning tree of G which has a minimum number of leaves subject to the constraint that v is a leaf (an internal vertex) in T. For convenience, given a tree T we denote by $\ell(T)$ the number of leaves in T. Given a set S, we denote by $S_1 \uplus S_2 = S$ a partition of S into two subsets S_1 and S_2.

2 The Problem on General Graphs

To decide the existence of a Hamiltonian path, Held and Karp used a dynamic programming recurrence [11]. This well-known technique is a very natural approach to attack the Maximum Internal Spanning Tree problem. In this section, we start by giving a Dynamic Programming based algorithm to compute a spanning tree with a maximum number of internal nodes in $\mathcal{O}^*(3^n)$ time and $\mathcal{O}^*(2^n)$ space. Our approach is different from the one given in [8] which requires $\mathcal{O}^*(3^n)$ space. In Subsection 2.2 we show how to speed up the running time to $\mathcal{O}^*(2^n)$ by using a fast evaluation algorithm for the subset convolution from [3]. We present the results for degree-bounded graphs in Subsection 2.3.

2.1 A dynamic programming approach

For the sake of simplicity, rather than computing a MIST of a given graph $G = (V, E)$, our algorithm computes the minimum number of leaves in a spanning tree of G. By standard back-

tracking techniques, it is easy to modify the algorithm so that it indeed returns a spanning tree with this number of leaves.

For every subset $S \subseteq V$ on at least two vertices and any vertex $v \in S$ we define $\mathsf{Opt}^L[S, v]$ (respectively $\mathsf{Opt}^I[S, v]$) as the minimum number of leaves in a spanning tree of $G[S]$ in which v is a leaf (respectively, in which v is an internal node), if one exists. In other words, $\mathsf{Opt}^L[S, v]$ (respectively, $\mathsf{Opt}^I[S, v]$) is the minimum number of leaves in any (v^L)-cst (respectively, in any (v^I)-cst) of $G[S]$, if such a spanning tree exists. If there is no such spanning tree then the value of $\mathsf{Opt}^L[S, v]$ (respectively, of $\mathsf{Opt}^I[S, v]$) is set to ∞.

For the base case, consider any 2-vertex set $S = \{u, v\} \subseteq V$. Clearly, if u and v are adjacent, then $\mathsf{Opt}^L[S, u] = \mathsf{Opt}^L[S, v] = 2$ and $\mathsf{Opt}^I[S, u] = \mathsf{Opt}^I[S, v] = \infty$. Otherwise, $\mathsf{Opt}^L[S, u] = \mathsf{Opt}^L[S, v] = \mathsf{Opt}^I[S, u] = \mathsf{Opt}^I[S, v] = \infty$ since $G[\{u, v\}]$ is disconnected and has no spanning tree.

Then the algorithm considers the subsets $S \subseteq V$ with $|S| \geq 3$ by increasing cardinality. For any $v \in S$, the values of $\mathsf{Opt}^L[S, v]$ and $\mathsf{Opt}^I[S, v]$ are computed using the following dynamic programming recurrences.

– If $G[S]$ is connected, then

$$\mathsf{Opt}^L[S, v] = \min_{u \in N(v) \cap S} \left\{ \mathsf{Opt}^L[S \setminus \{v\}, u], \mathsf{Opt}^I[S \setminus \{v\}, u] + 1 \right\}$$

$$\mathsf{Opt}^I[S, v] = \min_{\substack{(S_1 - v) \uplus (S_2 - v) = S - v \\ v \in S_1, v \in S_2 \\ |S_1|, |S_2| \geq 2}} \begin{cases} \mathsf{Opt}^I[S_1, v] + \mathsf{Opt}^I[S_2, v], \\ \mathsf{Opt}^I[S_1, v] + \mathsf{Opt}^L[S_2, v] - 1, \\ \mathsf{Opt}^L[S_1, v] + \mathsf{Opt}^L[S_2, v] - 2 \end{cases}$$

– Otherwise, $\mathsf{Opt}^L[S, v] = \infty$ and $\mathsf{Opt}^I[S, v] = \infty$.

Consider a vertex $u \in V$. Clearly, any optimum solution T of the MIST problem is either a (u^L)-cst or a (u^I)-cst with a minimum number of leaves. Thus, to obtain the minimum number of leaves in any spanning tree of a given graph $G = (V, E)$, it is sufficient to compute the value of $\mathsf{Opt}^L[V, v]$ and $\mathsf{Opt}^I[V, v]$ for some vertex $v \in V$. It remains to show that the formulae used by the dynamic programming approach are correct.

Lemma 1. *Let $G = (V, E)$ be a connected graph and let $v \in V$ such that G has a (v^L)-cst. There is a (v^L)-cst T^+ of G with a minimum number of leaves such that $T = T^+ \setminus \{v\}$ is a spanning tree of $G - v$ with a minimum number of leaves. In addition, denoting by u the neighbor of v in T, if T is a (u^L)-cst then $\ell(T^+) = \ell(T)$ and if T is a (u^I)-cst then $\ell(T^+) = \ell(T) + 1$.*

Proof. For the sake of contradiction, suppose that for every (v^L)-cst of G with a minimum number of leaves, the removal of v gives a spanning tree of $G - v$ which does not have a minimum number of leaves. Let T' be a (v^L)-cst of G with a minimum number of leaves, and suppose there exists a spanning tree T_1 of $G - v$ such that $\ell(T_1) < \ell(T' - v)$. Let u be the neighbor of v in T'. Construct the (v^L)-cst T^* of G obtained from T_1 by adding the edge $\{u, v\}$. Now, T^* is a (v^L)-cst of G with a minimum number of leaves because $\ell(T^*) \leq \ell(T_1) + 1 \leq \ell(T')$, and $T^* - v$ is a spanning tree of $G - v$ with a minimum number of leaves, a contradiction.

Additionally, if u is a leaf (resp. an internal node) of T, by adding v and the edge $\{u, v\}$ to T, we obtain the tree T^+ and the relation $\ell(T^+) = \ell(T)$ holds since u becomes internal in T^+ and v is a leaf of T^+ (resp. $\ell(T^+) = \ell(T) + 1$ since u is internal in T and in T^+). □

Lemma 2. *Let $G = (V, E)$ be a connected graph and let $v \in V$ such that G has a (v^I)-cst. There exists a (v^I)-cst tree T^+ with a minimum number of leaves and a partition $(V_1 - v) \uplus (V_2 - v)$ of $V - v$ where $v \in V_1, V_2$ such that $T_1 = T^+[V_1]$ and $T_2 = T^+[V_2]$ are spanning trees of $G[V_1]$ and $G[V_2]$ with a minimum number of leaves. In addition, if v is a leaf in both T_1 and T_2, then $\ell(T^+) = \ell(T_1) + \ell(T_2) - 2$, if v is a leaf in precisely one of T_1 and T_2, then $\ell(T^+) = \ell(T_1) + \ell(T_2) - 1$, and if v is internal in both T_1 and T_2, then $\ell(T^+) = \ell(T_1) + \ell(T_2)$.*

Proof. For the sake of contradiction, suppose that for every (v^I)-cst T of G with a minimum number of leaves, $T[V_1]$ is not a spanning tree with a minimum number of leaves of $G[V_1]$ or $T[V_2]$ is not a spanning tree with a minimum number of leaves of $G[V_2]$, where $V_1 - v$ and $V_2 - v$ are vertex sets of one or more connected components of $T - v$ that partition $V - v$ and $v \in V_1, V_2$. Let T' be a (v^I)-cst of G with a minimum number of leaves, and suppose there exists a spanning tree T_1 of $G[V_1]$ such that $\ell(T_1) < \ell(T'[V_1])$, where V_1 contains v and the vertex set of one or more (but not all) connected components of $T' - v$. Let $V_2 = (V \setminus V_1) \cup \{v\}$. Construct the (v^I)-cst T^* of G obtained by setting $T^* = T_1 \cup T'[V_2]$. We have $\ell(T^*) \leq \ell(T')$ and T^* is such that $T^*[V_1]$ has a minimum number of leaves. By using the same argument for V_2, we obtain a contradiction.

In addition, by identifying the vertex v in both trees T_1 and T_2 and merging the two trees at node v we obtain the tree T^+. This spanning tree T^+ has $\ell(T^+) = \ell(T_2) + \ell(T_2) - i$ leaves where $i \in \{0, 1, 2\}$ is the number of trees among T_1 and T_2 in which v is a leaf (since v becomes an internal node in T^+). □

We are now ready to establish the running time of our algorithm.

Theorem 1. *The given algorithm correctly computes a spanning tree with a minimum number of leaves in time $\mathcal{O}^*(3^n)$ and space $\mathcal{O}^*(2^n)$.*

Proof. By Lemmata 1 and 2, the dynamic programming recurrences are correct. To obtain a spanning tree of a given graph $G = (V, E)$ with a minimum number of leaves, it is sufficient to pick an arbitrary vertex v of V and to return the tree with fewer leaves among a (v^I)-cst and a (v^L)-cst of G with a minimum number of leaves. Thus the value $\min\{\mathsf{Opt}^L[V, v], \mathsf{Opt}^I[V, v]\}$ is the minimum number of leaves of any spanning tree of G.

The running time of the algorithm is $\mathcal{O}^*(3^n)$ as the algorithm goes through all partitions of V into $V \setminus S$, $S_1 - v$, $S_2 - v$, $\{v\}$. It is easy to see that the needed space is $\mathcal{O}^*(2^n)$. □

2.2 Speed-up by subset convolution

Motivated by Held's and Karp's algorithm for Hamiltonian Path [11], the problem of solving the MIST problem in time $\mathcal{O}^*(2^n)$ is an intriguing question. This question was settled by Nederlof in [17] who provides an Inclusion-Exclusion based algorithm working in $\mathcal{O}^*(2^n)$ time. In this section we also provide an $\mathcal{O}^*(2^n)$ time algorithm using fast subset convolution. Whereas the approach of Nederlof needs only polynomial space, subset convolution takes exponential space. However, our approach extends to degree-bounded graphs for which a faster running time is obtained in Subsection 2.3. Neither the approach of Nederlof [17] nor the approach of Fomin et al. [10] seem to be able to beat a running time of $\mathcal{O}^*(2^n)$ for degree-bounded graphs.

In [3], Björklund et al. give a fast algorithm for the subset convolution problem. Given two functions f and g, their subset convolution $f * g$ is defined for all $S \subseteq V$ by $(f * g)(S) = \sum_{T \subseteq S} f(T)g(S \setminus T)$. They show that via Möbius transform and inversion, the subset convolution can be computed in $\mathcal{O}^*(2^n)$ time, improving on the trivial $\mathcal{O}^*(3^n)$-time algorithm. They exemplify the technique by speeding-up the Dreyfus-Wagner Steiner tree algorithm to $\mathcal{O}^*(2^k n^2 M + nm \log M)$ where k is the number of terminals and M the maximum weight over all edges.

Let us explain how subset convolution can be used to speed-up the evaluation of the recursion of the previous section. We apply the fast subset convolution over the min-sum semiring:

$$(f * g)(S) = \min_{T \subseteq S}\{f(T) + g(S \setminus T)\}.$$

As Björklund et al. noticed, for such recurrences the computation has to be done in a level-wise manner. In our case, the computation of $\mathsf{Opt}^I[S, v]$ requires the values of the already computed $\mathsf{Opt}^I[X, v]$ for all sets $X \subset S$ of size at least 2 containing v.

At each level l, $3 \leq l \leq n$, assume that $\mathsf{Opt}^L[X,x]$ and $\mathsf{Opt}^I[X,x]$ have already been computed for all $X \subseteq V$ with $2 \leq |X| \leq l-1$ and $x \in X$. At level l, we compute the values of $\mathsf{Opt}^I[X,x]$ and the values of $\mathsf{Opt}^L[X,x]$ for all $X \subseteq V$ and $x \in X$ with $|X| = l$. Recall that the computation of $\mathsf{Opt}^L[X,x]$ and $\mathsf{Opt}^I[X,x]$ for all $|X| \leq 2$ is *easy* and can be done in $\mathcal{O}(n^2)$ time (see Section 2.1).

Assume that $X \subseteq V$, $|X| = \ell \geq 3$, $x \in X$, and that all values of Opt^L and Opt^I have already been computed for all previous levels. Note that, for each X and x, the values of $\mathsf{Opt}^L[X,x]$ can be computed in polynomial time using the recurrence of the previous subsection. To compute $\mathsf{Opt}^I[X,x]$ we define the functions $f_x(Y,l)$ and $g_x(Y,l)$ for all subsets Y of X.

$$f_x(Y,l) = \begin{cases} \mathsf{Opt}^L[Y \cup \{x\}, x] & \text{if } 1 \leq |Y| \leq l-2, \textit{and} \\ \infty & \text{otherwise.} \end{cases}$$

$$g_x(Y,l) = \begin{cases} \mathsf{Opt}^I[Y \cup \{x\}, x] & \text{if } 1 \leq |Y| \leq l-2, \textit{and} \\ \infty & \text{otherwise.} \end{cases}$$

Then we define the three functions:

$$h_x^{II}(X,l) = (g_x * g_x)(X-x,l),$$
$$h_x^{IL}(X,l) = (f_x * g_x)(X-x,l),$$
$$h_x^{LL}(X,l) = (f_x * f_x)(X-x,l).$$

These functions can be evaluated for each level l via subset convolution over the min-sum semiring in total time $\mathcal{O}^*(2^n)$ (see Theorem 3 in [3]). Note that these functions were derived from the recurrence established in Subsection 2.1 to compute $\mathsf{Opt}^I[X,x]$. In particular, to compute $\mathsf{Opt}^I[X,x]$ we need to look at each partition $(X_1 - x) \uplus (X_2 - x) = (X - x)$ such that $x \in X_1 \cap X_2$ and $|X_1|, |X_2| \geq 2$. Thus at level l, only the values of $f_x(Y,l)$ and $g_x(Y,l)$ with $|Y| < l-1$ are of interest (and the ones with $|Y| = l-1$ are set to ∞). Finally the value of $\mathsf{Opt}^I[X,x]$ is set to $\min\{h_x^{II}(X,l), h_x^{IL}(X,l)-1, h_x^{LL}(X,l)-2\}$.

Theorem 2. *The algorithm computes a spanning tree with a minimum number of leaves in time and space $\mathcal{O}^*(2^n)$.*

2.3 A consequence for graphs of bounded degree

In this section, we extend the presented results to degree-bounded graphs. In [4], Björklund et al. show that the number of connected vertex sets is smaller than 2^n in graphs with bounded degree.

Lemma 3 (Lemma 3 in [4]). *An n-vertex graph with maximum vertex degree Δ has at most $\beta_\Delta^n + n$ connected vertex sets with $\beta_\Delta = (2^{\Delta+1} - 1)^{1/(\Delta+1)}$.*

By combining this lemma and the fast subset convolution of [3], one can show that it is sufficient to go only through the $\mathcal{O}^*(\beta_\Delta^n)$ connected subsets X for evaluating the convolution. Björklund et al. showed this already for the Travelling Salesman Problem. Indeed, one can observe that any subset $X \subseteq V$ such that $G[X]$ is not connected would introduce some (useless) ∞ terms (either with a positive or a negative sign).

This final observation implies the following result on graphs of bounded degree.

Theorem 3. *For any graph with maximum degree Δ, a spanning tree with a minimum number of leaves can be computed in time $\mathcal{O}^*(\beta_\Delta^n)$ with $\beta_\Delta = (2^{\Delta+1} - 1)^{1/(\Delta+1)}$ and exponential-space.*

Some concrete figures for the corresponding bases of the exponential functions are listed in Table 1.

Δ	3	4	5	6
running times	$\mathcal{O}(1.9680^n)$	$\mathcal{O}(1.9874^n)$	$\mathcal{O}(1.9948^n)$	$\mathcal{O}(1.9978^n)$

Table 1. Running time of the algorithm according to the maximum degree Δ.

3 Subcubic Maximum Internal Spanning Tree

In this section, we focus on graphs with maximum degree at most three. We start with some interesting observations and then we present the reduction rules used by our algorithm. Then the algorithm is described and its running time is established. Namely, we establish an $\mathcal{O}(1.8669^n)$-time algorithm and provide a lower bound on its worst-case running time. Finally a parameterized algorithm is given and we establish an $\mathcal{O}(2.1364^k n^{\mathcal{O}(1)})$ running time thanks to a *Measure & Conquer* approach.

3.1 Observations

Let t_i^T denote the number of vertices u such that $d_T(u) = i$ for a spanning tree T. Then the following proposition can be proved by induction on $n_T := |V(T)|$.

Proposition 1. *In any spanning tree T, $2 + \sum_{i\geq 3}(i-2)\cdot t_i^T = t_1^T$.*

Due to Proposition 1, MIST on subcubic graphs boils down to finding a spanning tree T such that t_2^T is maximum. Every internal vertex of higher degree would also introduce additional leaves.

Lemma 4. *[19] An optimal solution T_o to* Max Internal Spanning Tree *is a Hamiltonian path or the leaves of T_o are independent.*

The proof of Lemma 4 shows that if T_o is not a Hamiltonian path and there are two adjacent leaves, then the number of internal vertices can be increased. In the rest of the paper we assume that T_o is not a Hamiltonian path due to the next lemma.

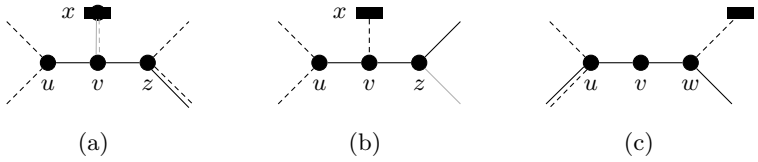
Lemma 5. Hamiltonian Path *can be solved in time $\mathcal{O}(1.251^n)$ on subcubic graphs.*

Proof. Let $G = (V, E)$ be a subcubic graph. Run the algorithm of [12] to find a Hamiltonian cycle. If it succeeds G clearly also has a Hamiltonian path. If it does not succeed we have to investigate whether G has a Hamiltonian path whose end points are not adjacent. Let $u, v \in V(G)$ be two non-adjacent vertices. To check whether G has a Hamiltonian path uPv, we check whether $G' = (V, E')$, with $E' := E \cup \{\{u, v\}\}$ has a Hamiltonian cycle. If G' has maximum degree at most 3, then run the algorithm of [12]. Otherwise, choose a vertex of degree 4, say u, and two neighbors x, z of u distinct from v. As $\{u, v\}$ belongs to every existent Hamiltonian cycle of G' (otherwise G has a Hamiltonian cycle as well), every Hamiltonian cycle of G' avoids $\{u, x\}$ or $\{u, z\}$. Recursively check whether $(V, E' \setminus \{\{u, x\}\})$ or $(V, E' \setminus \{\{u, z\}\})$ has a Hamiltonian cycle. This recursion has depth at most 2 since G' has at most 2 vertices of degree 4. The Hamiltonian Cycle algorithm of [12] is executed at most $4(n(n-1)/2 - m)$ times. This algorithms runs in $\mathcal{O}^*(2^{(31/96)n}) \subseteq \mathcal{O}^*(1.2509^n)$ steps. □

At this point we prove an auxiliary lemma used for the analysis of the forthcoming algorithm.

Lemma 6. *Let $G = (V, E)$ be a graph and let T be a spanning tree and $u, v \in V(T)$ two adjacent vertices with $d_T(u) = d_T(v) = 3$ such that $\{u, v\}$ is not a bridge in G. Then there is a spanning tree $T' \supset (T \setminus \{\{u, v\}\})$ with $|internal(T')| \geq |internal(T)|$ and $d_{T'}(u) = d_{T'}(v) = 2$.*

Proof. By removing $\{u, v\}$, T is separated into two parts T_1 and T_2. The vertices u and v become 2-vertices. As $\{u, v\}$ is not a bridge, there is another edge $e \in E \setminus T$ connecting T_1 and T_2. By adding e we lose at most two 2-vertices. Then let $T' := (T \setminus \{\{u, v\}\}) \cup \{e\}$ and it follows that $|internal(T')| \geq |internal(T)|$. □



(a) (b) (c)

Fig. 1. Local configurations to which the reduction rules **Attach** and **Special** apply, respectively. The following drawing conventions apply to all figures of this article. Light gray edges may be not present. Dashed edges are in F, solid edges are in $E \setminus F$ and double edges (solid and dashed) are in E (possibly in F, but not necessarily). Edges incident to oblongs are pt-edges and the oblong represents the vertex of degree one in this case. In Fig. 1(a), vx might be a pt-edge, an edge in $E \setminus F$, or a non-edge.

3.2 Reduction Rules

Let $E' \subseteq E$. Then, $\partial E' := \{\{u, v\} \in E \setminus E' \mid u \in V(E')\}$ are the edges outside E' that have a common end point with an edge in E' and $\partial_V E' := V(\partial E') \cap V(E')$ are the vertices that have at least one incident edge in E' and another incident edge not in E'. In the course of the algorithm we will maintain an acyclic subset of edges F which will be part of the final solution. The following invariant will always be true: $G[F]$ consists of a tree T and a set P of *pending tree edges (pt-edges)*. Here a pt-edge $\{u, v\} \in F$ is an edge with one end point u of degree 1 and the other end point $v \notin V(T)$, see Figure 1(b) where $\{x, v\}$ is a pt-edge. $G[T \cup P]$ will always consist of $1 + |P|$ components. Occasionally, double edges (i.e., two edges connecting the same pair of vertices) will appear during the execution of the algorithm. However, they are instantly removed by a reduction rule (rule **DoubleEdge** below), so that we may otherwise assume that G is a simple graph.

Next we present a sequence of reduction rules. Note that the order in which they are applied is crucial. We assume that before a rule is applied the preceding ones were carried out exhaustively.

Bridge: If there is a bridge $e \in \partial E(T)$, then add e to F.

DoubleEdge: If there is a double edge delete one of them which is not in F.

Cycle: Delete any edge $e \in E$ such that $T \cup \{e\}$ has a cycle.

Deg1: Let $u \in V \setminus V(F)$ with $d(u) = 1$. Then add its incident edge to F.

Pending: If there is a vertex v that is incident to $d_G(v) - 1$ pt-edges, then remove its incident pt-edges.

ConsDeg2: If there are edges $\{v, w\}, \{w, z\} \in E \setminus T$ such that $d_G(w) = d_G(z) = 2$, then delete $\{v, w\}, \{w, z\}$ from G and add the edge $\{v, z\}$ to G.

Deg2: If there is an edge $\{u, v\} \in \partial E(T)$ such that $u \in V(T)$ and $d_G(u) = 2$, then add $\{u, v\}$ to F.

Attach: If there are edges $\{u, v\}, \{v, z\} \in \partial E(T)$ such that $u, z \in V(T)$, $d_T(u) = 2$, $1 \leq d_T(z) \leq 2$, then delete $\{u, v\}$; see Fig. 1(a).

Attach2: If there is a vertex $u \in \partial_V E(T)$ with $d_T(u) = 2$ and $\{u, v\} \in E \setminus T$ such that v is incident to a pt-edge, then delete $\{u, v\}$; see Fig. 1(b).

Special: If there are two edges $\{u, v\}, \{v, w\} \in E \setminus F$ with $d_T(u) \geq 1$, $d_G(v) = 2$, and w is incident to a pt-edge, then add $\{u, v\}$ to F; see Fig. 1(c).

We mention that **ConsDeg2** is the only redution rule which can create double edges. In this case **DoubleEdge** will delete one of them which is not in F. It will be assured by the reduction rules and the forthcoming algorithm that at most one can be part of F.

Lemma 7. *The reduction rules stated above are sound.*

Proof. Let $T_o \supset F$ be a spanning tree of G with a maximum number of internal vertices. The first four rules are correct for the purpose of connectedness and acyclicity of the evolving spanning tree.

Pending is correct as the other edge incident to v (which will be added to P by a subsequent **Deg1** rule) is a bridge and needs to be in any spanning tree.

ConsDeg2 Let G' be the graph after the reduction rule was applied. We implicitly assume that we can add $\{w, z\}$ to $T_o \supset F$ which is an optimal solution for G. If $\{w, z\} \notin T_o$ then $\{v, w\} \in T_o$. Then we can simply exchange the two edges giving a solution $\tilde{T}_o$ with $\{w, z\} \in \tilde{T}_o$ and $t_2^{T_o} \leq t_2^{\tilde{T}_o}$. By contracting the edge $\{w, z\}$ we receive a solution T'_o such that $|internal(T'_o)| = |internal(T_o)| - 1$.
Now suppose T'_o is a spanning tree for G'. If $\{v, z\} \in T'_o$ then let $T_o = T'_o \setminus \{\{v, z\}\} \cup \{\{v, w\}, \{w, z\}\}$. T_o is a spanning tree for G and we have $|internal(T_o)| = |internal(T'_o)| + 1$. If $\{v, z\} \notin T'_o$ then by connectivity $d_{T'_o}(z) = 1$. Let $T_o = T'_o \cup \{\{v, w\}\}$ the $|internal(T_o)| = |internal(T'_o)| + 1$.

Deg2 Since the preceding reduction rules do not apply, we have $d_G(v) = 3$. Assume u is a leaf in T_o. There is exactly one incident edge, say $\{v, z\}$, $z \neq u$, that is not pending such that it is contained in the single cycle in $G[T \cup \{\{u, v\}\}]$. Define another spanning tree $T'_o \supset F$ by setting $T'_o = (T_o \cup \{\{u, v\}\}) \setminus \{v, z\}$. Since $|internal(T_o)| \leq |internal(T'_o)|$, T'_o is also optimal.

Attach If $\{u, v\} \in T_o$ then $\{v, z\} \notin T_o$ due to the acyclicity of T_o and as T_o is connected. Then by exchanging $\{u, v\}$ and $\{v, z\}$ we obtain a solution T'_o with at least as many 2-vertices.

Attach2 Suppose $\{u, v\} \in T_o$. Let $\{v, p\}$ be the pt-edge and $\{v, z\}$ the third edge incident to v (that must exist and is not pending, since **Pending** did not apply). Since **Bridge** did not apply, $\{u, v\}$ is not a bridge. Firstly, suppose $\{v, z\} \in T_o$. Due to Lemma 6, there is also an optimal solution $T'_o \supset F$ with $\{u, v\} \notin T'_o$. Secondly, assume $\{v, z\} \notin T_o$. Then $T' = (T_o \setminus \{\{u, v\}\}) \cup \{\{v, z\}\}$ is also optimal as u has become a 2-vertex.

Special Suppose $\{u, v\} \notin T_o$. Then $\{v, w\}, \{w, z\} \in T_o$ where $\{w, z\}$ is the third edge incident to w. Let $T'_o := (T_o \setminus \{\{v, w\}\}) \cup \{\{u, v\}\}$. In T'_o, w is a 2-vertex and hence T'_o is also optimal. □

3.3 The Algorithm

The algorithm we describe here is recursive. It constructs a set F of edges which are selected to be in every spanning tree considered in the current recursive step. The algorithm chooses edges and considers all relevant choices for adding them to F or removing them from G. It selects these edges based on priorities chosen to optimize the running time analysis. Moreover, the set F of edges will always be the union of a tree T and a set of edges P that are not incident to the tree and have one end point of degree 1 in G (pt-edges). We do not explicitly write in the algorithm that edges move from P to T whenever an edge is added to F that is incident to both an edge of T and an edge of P. To maintain the connectivity of T, the algorithm explores edges in the set $\partial E(T)$ to grow T.

If $|V| > 2$ every spanning tree T must have a vertex v with $d_T(v) \geq 2$. Thus initially the algorithm creates an instance for every vertex v and every possibility that $d_T(v) \geq 2$. Due to the degree constraint there are no more than $4n$ instances. After this initial phase, the algorithm proceeds as as follows.

1. Carry out each reduction rule exhaustively in the given order (until no rule applies).
2. If $\partial E(T) = \emptyset$ and $V \neq V(T)$, then G is not connected and does not admit a spanning tree. Ignore this branch.
3. If $\partial E(T) = \emptyset$ and $V = V(T)$, then return T.
4. Select $\{a, b\} \in \partial E(T)$ with $a \in V(T)$ according to the following priorities (if such an edge exists):

a) there is an edge $\{b, c\} \in \partial E(T)$,
b) $d_G(b) = 2$,
c) b is incident to a pt-edge, or
d) $d_T(a) = 1$.

Recursively solve the two instances where $\{a, b\}$ is added to F or removed from G respectively, and return the spanning tree with most internal vertices.

5. Otherwise, select $\{a, b\} \in \partial E(T)$ with $a \in V(T)$. Let c, x be the other two neighbors of b. Recursively solve three instances where
   (i) $\{a, b\}$ is removed from G,
   (ii) $\{a, b\}$ and $\{b, c\}$ are added to F and $\{b, x\}$ is removed from G, and
   (iii) $\{a, b\}$ and $\{b, x\}$ are added to F and $\{b, c\}$ is removed from G.
   Return the spanning tree with most internal vertices.

3.4 An Exact Analysis of the Algorithm

By a *Measure & Conquer* analysis taking into account the degrees of the vertices, their number of incident edges that are in F, and to some extent the degrees of their neighbors, we obtain the following result.

Theorem 4. MIST *can be solved in time* $\mathcal{O}(1.8669^n)$ *on subcubic graphs.*

Let $D_2 := \{v \in V \mid d_G(v) = 2, d_F(v) = 0\}$, $D_3^\ell := \{v \in V \mid d_G(v) = 3, d_F(v) = \ell\}$ and $D_3^{2*} := \{v \in D_3^2 \mid N_G(v) \setminus N_F(v) = \{u\} \text{ and } d_G(u) = 2\}$. Then the measure we use for our running time bound is

$$\mu(G) = \omega_2 \cdot |D_2| + |D_3^0| + \omega_3^1 \cdot |D_3^1| + \omega_3^2 \cdot |D_3^2 \setminus D_3^{2*}| + \omega_3^{2*} \cdot |D_3^{2*}|$$

with the weights $\omega_2 = 0.3193$, $\omega_3^1 = 0.6234$, $\omega_3^2 = 0.3094$ and $\omega_3^{2*} = 0.4144$.

Let $\Delta_3^0 := \Delta_3^{0*} := 1 - \omega_3^1$, $\Delta_3^1 := \omega_3^1 - \omega_3^2$, $\Delta_3^{1*} := \omega_3^1 - \omega_3^{2*}$, $\Delta_3^2 := \omega_3^2$, $\Delta_3^{2*} := \omega_3^{2*}$ and $\Delta_2 = 1 - \omega_2$. We define $\tilde{\Delta}_3^i := \min\{\Delta_3^i, \Delta_3^{i*}\}$ for $1 \leq i \leq 2$, $\Delta_m^\ell = \min_{0 \leq j \leq \ell}\{\Delta_3^j\}$, and $\tilde{\Delta}_m^\ell = \min_{0 \leq j \leq \ell}\{\tilde{\Delta}_3^j\}$.
The proof of the theorem uses the following result.

Lemma 8. *None of the reduction rules increase* μ *for the given weights.*

Proof. **Bridge**, **Deg1**, **Deg2** and **Special** add edges to F. Due to the definitions of D_3^ℓ and D_3^{2*} and the choice of the weights it can be seen that the addition of an edge to F can only decrease μ. It is also easy to see that the deletion of edges $\{u, v\}$ with $d_T(u) \geq 1$ is safe with respect to u: the weight of u can only decrease due to this. Nevertheless, the rules which delete edges might cause that a $v \in D_3^2 \setminus D_3^{2*}$ will be in D_3^{2*} afterwards. Thus, we have to prove that in this case the overall reduction is enough. A vertex $v \in D_3^2 \setminus D_3^{2*}$ moves to D_3^{2*} through the deletion of an edge $\{x, y\}$ if x (or y) has degree 3, $x \notin \partial_V(T)$ and $\{v, x\} \in E \setminus F$. If x were in $\partial_V(T)$, the reduction rule **Cycle** would remove the edge $\{v, x\}$ and the weight of v would drop to 0. Thus, only the appearance of a degree-2 vertex x with $x \notin \partial_V(T)$ may cause a vertex to move from $D_3^2 \setminus D_3^{2*}$ to D_3^{2*}. The next reduction rule which may create vertices of degree 2 is **Attach** when $d(v) = 3$ (where v is as in the rule definition). If $d_T(z) = 2$, then z moves from $D_3^2 \setminus D_3^{2*}$ to D_3^{2*} through the deletion of the edge $\{u, v\}$. However, the total reduction of μ through the application of this reduction rule is at least $\omega_3^2 + \Delta_2 - (\omega_3^{2*} - \omega_3^2) > 0$. It can be checked that no other reduction rule creates degree-2 vertices not already contained in $\partial_V(T)$. □

Proof. (of Theorem 4) As the algorithm deletes edges or moves edges from $E \setminus F$ to F, cases 1–3 do not contribute to the exponential function in the running time of the algorithm. It remains to analyze cases 4 and 5, which we do now. Note that after applying the reduction rules exhaustively, we have that for all $v \in \partial_V E(T)$, $d_G(v) = 3$ (**Deg2**) and for all $u \in V$, $d_P(u) \leq 1$ (**Pending**).

4.(a) Obviously, $\{a,b\},\{b,c\} \in E \setminus T$, and there is a vertex d such that $\{c,d\} \in T$; see Figure 2(a). We have $d_T(a) = d_T(c) = 1$ due to the reduction rule **Attach**. We consider three cases.

$d_G(b) = 2$. When $\{a,b\}$ is added to F, **Cycle** deletes $\{b,c\}$. We get an amount of $\omega_2 + \omega_3^1$, as b drops out of D_2 and c out of D_3^1 (**Deg2**). Also a will be removed from D_3^1 and added to D_3^2 which amounts to a reduction of at least $\tilde{\Delta}_3^1$. When $\{a,b\}$ is deleted, $\{b,c\}$ is added to T (**Bridge**). By a symmetric argument we get a reduction of $\omega_2 + \omega_3^1 + \tilde{\Delta}_3^1$ as well. In total this yields a $(\omega_2 + \omega_3^1 + \tilde{\Delta}_3^1, \omega_2 + \omega_3^1 + \tilde{\Delta}_3^1)$-branch.

$d_G(b) = 3$ and there is one pt-edge incident to b. Adding $\{a,b\}$ to F decreases the measure by $\tilde{\Delta}_3^1$ (from a) and $2\omega_3^1$ (deleting $\{b,c\}$, then **Deg2** on c). By Deleting $\{a,b\}$ we decrease μ by $2\omega_3^1$ and by $\tilde{\Delta}_3^1$ (from c). This amounts to a $(2\omega_3^1 + \tilde{\Delta}_3^1, 2\omega_3^1 + \tilde{\Delta}_3^1)$-branch.

$d_G(b) = 3$ and no pt-edge is incident to b. Let $\{b,z\}$ be the third edge incident to b. In the first branch the measure drops by at least $\omega_3^1 + \tilde{\Delta}_3^1$ from c and a (**Deg2**), 1 from b (**Deg2**). In the second branch we get $\omega_3^1 + \Delta_2$. Observe that we also get an amount of at least $\tilde{\Delta}_m^1$ from $q \in N_T(a) \setminus \{b\}$ if $d_G(q) = 3$. If $d_G(q) = 2$ we get ω_2. It results a $(\omega_3^1 + \tilde{\Delta}_3^1 + 1, \omega_3^1 + \Delta_2 + \min\{\omega_2, \tilde{\Delta}_m^1\})$-branch.

Note that from this point on, for all $u,v \in V(T)$ there is no $z \in V \setminus V(T)$ with $\{u,z\},\{z,v\} \in E \setminus T$.

4.(b) As the previous case does not apply, the other neighbor c of b has $d_T(c) = 0$, and $d_G(c) \geq 2$ (**Pending**). Additionally, observe that $d_G(c) = 3$ due to **ConsDeg2** and that $d_P(c) = 0$ due to **Special**, see Figure 2(b). We consider two subcases.

$d_T(a) = 1$. When we add $\{a,b\}$ to F, then $\{b,c\}$ is also added due to **Deg2**. The reduction is at least $\tilde{\Delta}_3^1$ from a, ω_2 from b and Δ_3^0 from c. When $\{a,b\}$ is deleted, $\{b,c\}$ becomes a pt-edge. There is $\{a,z\} \in E \setminus T$ with $z \neq b$, which is subject to a **Deg2** reduction rule. We get at least ω_3^1 from a, ω_2 from b, Δ_3^0 from c and $\min\{\omega_2, \tilde{\Delta}_m^1\}$ from z. This is a $(\tilde{\Delta}_3^1 + \Delta_3^0 + \omega_2, \omega_3^1 + \Delta_3^0 + \omega_2 + \min\{\omega_2, \tilde{\Delta}_m^1\})$-branch.

$d_T(a) = 2$. Similarly, we obtain a $(\Delta_3^{2*} + \omega_2 + \Delta_3^0, \Delta_3^{2*} + \omega_2 + \Delta_3^0)$-branch.

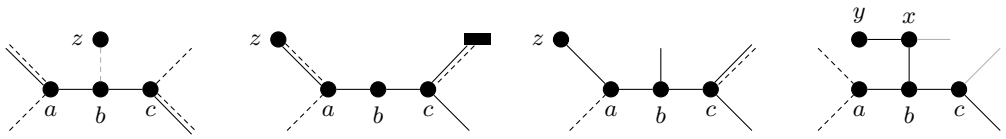
4.(c) In this case, $d_G(b) = 3$ and there is one pt-edge attached to b, see Figure 2(c). Note that $d_T(a) = 2$ can be ruled out due to **Attach2**. Thus, $d_T(a) = 1$. Let $z \neq b$ be such that $\{a,z\} \in E \setminus T$. Due to the priorities, $d_G(z) = 3$. We distinguish between the cases where c, the other neighbor of b, is incident to a pt-edge or not.

$d_P(c) = 0$. First suppose $d_G(c) = 3$. Adding $\{a,b\}$ to F allows a reduction of $2\Delta_3^1$ (due to case 4.(b) we can exclude Δ_3^{1*}). Deleting $\{a,b\}$ implies that we get a reduction from a and b of $2\omega_3^1$ (**Deg2** and **Pending**). As $\{a,z\}$ is added to F we reduce $\mu(G)$ by at least $\tilde{\Delta}_m^1$ as the state of z changes. Now due to **Pending** and **Deg1** we include $\{b,c\}$ and get Δ_3^0 from c. We have at least a $(2\Delta_3^1, 2\omega_3^1 + \tilde{\Delta}_m^1 + \Delta_3^0)$-branch.

If $d_G(c) = 2$ we consider the two cases for z also. These are $d_P(z) = 1$ and $d_P(z) = 0$. The first entails $(\omega_3^1 + \Delta_3^{1*}, 2\omega_3^1 + \tilde{\Delta}_3^1 + \omega_2 + \tilde{\Delta}_m^2)$. Note that when we add $\{a,b\}$ we trigger **Attach2** and by deleting $\{a,b\}$ all edges incident to c become bridges. The second is a $(\Delta_3^1 + \Delta_3^{1*}, 2\omega_3^1 + \Delta_3^0 + \omega_2 + \tilde{\Delta}_m^2)$-branch.

$d_P(c) = 1$. Let $d \neq b$ be the other neighbor of c that does not have degree 1. When $\{a,b\}$ is added to F, $\{b,c\}$ is deleted by **Attach2** and $\{c,d\}$ becomes a pt-edge (**Pending** and **Deg1**). The changes on a incur a measure decrease of Δ_3^{1*} and those on b,c a measure decrease of $2\omega_3^1$. When $\{a,b\}$ is deleted, $\{a,z\}$ is added to F (**Deg2**) and $\{c,d\}$ becomes a pt-edge by two applications of the **Pending** and **Deg1** rules. Thus, the decrease of the measure is at least $3\omega_3^1$ in this branch. In total, we have a $(\Delta_3^{1*} + 2\omega_3^1, 3\omega_3^1)$-branch here.

4.(d) Now, $d_G(b) = 3$, b is not incident to a pt-edge, and $d_T(a) = 1$. See Figure 2(c). There is also some $\{a,z\} \in E \setminus T$ such that $z \neq b$. Note that $d_T(z) = 0$, $d_G(z) = 3$ and $d_P(z) = 0$. Otherwise, either **Cycle** or cases 4.(b) or 4.(c) would have been triggered. From the addition of $\{a,b\}$ to F we get $\Delta_3^1 + \Delta_3^0$ and from its deletion ω_3^1 (from a via **Deg2**), Δ_2 (from b) and at least Δ_3^0 from z and thus, a $(\Delta_3^1 + \Delta_3^0, \omega_3^1 + \Delta_2 + \Delta_3^0)$-branch.



(a) (b) (c) (d)

Fig. 2. Local configurations corresponding to the different cases of the algorithm where a branching occurs.

5. See Figure 2(d). The algorithm branches in the following way: 1) Delete $\{a, b\}$, 2) add $\{a, b\}, \{b, c\}$, and delete $\{b, x\}$, 3) add $\{a, b\}, \{b, x\}$ and delete $\{b, c\}$. Observe that these cases are sufficient to preserve an optimal solution. Due to **Deg2**, we can disregard the case when b is a leaf. Due to Lemma 6 we also disregard the case when b is a 3-vertex as $\{a, b\}$ is not a bridge. Thus by branching in this manner we find at least one optimal solution.

The reduction in the first branch is at least $\omega_3^2 + \Delta_2$. We get an additional amount of ω_2 if $d(x) = 2$ or $d(c) = 2$ from **ConsDeg2**. In the second branch we have to consider also the vertices c and x. We distinguish between three situations for $h \in \{c, x\}$ $\alpha)$ $d_G(h) = 2$, $\beta)$ $d_G(h) = 3$, $d_P(h) = 0$ and $\gamma)$ $d_G(h) = 3$, $d_P(h) = 1$. We will only analyze branch 2) as 3) is symmetric. We first get a reduction of $\omega_3^2 + 1$ from a and b. We reduce μ due to deleting $\{b, x\}$ by: $\alpha)$ $\omega_2 + \tilde{\Delta}_m^2$ by **Pending**, $\beta)$ Δ_2, $\gamma)$ $\omega_3^1 + \tilde{\Delta}_m^2$ by **Pending** and **Deg1**. Next we examine the amount by which μ will be decreased by adding $\{b, c\}$ to F. We distinguish between the cases α, β and γ: $\alpha)$ $\omega_2 + \tilde{\Delta}_m^2$, $\beta)$ Δ_3^0, $\gamma)$ $\tilde{\Delta}_3^1$.
For $h \in \{c, x\}$ and $W \in \{\alpha, \beta, \gamma\}$ let 1_W^h be the indicator function which is set to one if we have situation W at vertex h. Otherwise, it is zero. Now the branching tuple can be stated the following way :
$(\omega_3^2 + \Delta_2 + (1_\alpha^x + 1_\alpha^c) \cdot \omega_2,$
$\omega_3^2 + 1 + 1_\alpha^x \cdot (\omega_2 + \tilde{\Delta}_m^2) + 1_\beta^x \cdot \Delta_2 + 1_\gamma^x \cdot (\omega_3^1 + \tilde{\Delta}_m^2) + 1_\alpha^c \cdot (\omega_2 + \tilde{\Delta}_m^2) + 1_\beta^c \cdot \Delta_3^0 + 1_\gamma^c \cdot \tilde{\Delta}_3^1),$
$\omega_3^2 + 1 + 1_\alpha^c \cdot (\omega_2 + \tilde{\Delta}_m^2) + 1_\beta^c \cdot \Delta_2 + 1_\gamma^c \cdot (\omega_3^1 + \tilde{\Delta}_m^2) + 1_\alpha^x \cdot (\omega_2 + \tilde{\Delta}_m^2) + 1_\beta^x \cdot \Delta_3^0 + 1_\gamma^x \cdot \tilde{\Delta}_3^1)$
The amount of $(1_\alpha^x + 1_\alpha^c) \cdot \omega_2$ comes from possible applications of **ConsDeg2**.

Observe that every instance created by branching is smaller than the original instance in terms of μ. Together with Lemma 8 we see that every step of the algorithm only decreases μ. Now if we evaluate the upper bound for every given branching tuple for the given weights we can conclude that MIST can be solved in time $\mathcal{O}^*(1.8669^n)$ on subcubic graphs. □

Lower Bound. To complete the running time analysis — with respect to the number of vertices — of this algorithm, we provide a lower bound here, i.e., we exhibit an infinite family of graphs for which the algorithm requires $\Omega(c)$ steps, where $c = \sqrt[4]{2} > 1.1892$ and n is the number of vertices in the input graph.

Theorem 5. *There is an infinite family of graphs such that our algorithm for* MIST *needs* $\Omega(\sqrt[4]{2}^n)$ *time.*

Proof. Instead of giving a formal description of the graph family, consider Fig. 3.

Assume that the bold edges are already in F. Then the dark gray vertices are in $\partial_V(T)$ and all non-bold edges connecting two black vertices have been removed from the graph. Step 4.(d) of the algorithm selects an edge connecting one of the dark gray vertices with one of the light gray vertices; w.l.o.g., the algorithm selects the edge $\{a, b\}$. In the branch where $\{a, b\}$ is added to F, reduction rules **Cycle** and **Deg2** remove $\{d, b\}$ and $\{a, e\}$ from the graph and add

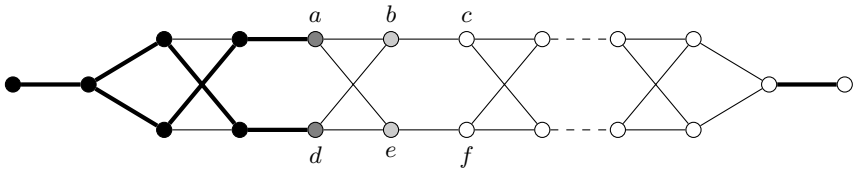


Fig. 3. Graph used to lower bound the running time of our MIST algorithm.

$\{\{d,e\},\{b,c\},\{e,f\}\}$ to F. In the branch where $\{a,b\}$ is removed from G, reduction rules **Cycle** and **Deg2** remove $\{d,e\}$ from the graph and add $\{\{a,e\},\{d,b\},\{b,c\},\{e,f\}\}$ to F. In either branch, 4 more vertices have become internal, the number of leaves has remained unchanged, and we arrive at a situation that is basically the same as the one we started with, so that the argument repeats. Hence, when we start with a graph with $n = 4h + 4$ vertices, then there will be a branching scenario that creates a binary search tree of height h. □

3.5 A Parameterized Analysis of the Algorithm

In this section, we are going to provide another analysis of our algorithm, this time from the viewpoint of parameterized complexity, where the parameter k is a lower bound on the size of the sought solution, i.e., we look for a spanning tree with at least k internal nodes. Notice that also the worst-case example from Theorem 5 carries over, but the according estimate of $\Omega(\sqrt[4]{2}^k)$ is less interesting.

Let us first investigate the possibility of running our algorithm on a linear vertex kernel, which offers the simplest way of using Measure & Conquer in parameterized algorithms. For general graphs, the smallest known kernel has at most $3k$ vertices [9]. This can be easily improved to $2k$ for subcubic graphs.

Lemma 9. *MIST on subcubic graphs has a $2k$-kernel.*

Proof. Compute an arbitrary spanning tree T. If it has at least k internal vertices, answer YES (in other words, output a small trivial YES-instance). Otherwise, $t_3^T + t_2^T < k$. Then, by Proposition 1, $t_1^T < k + 2$. Thus, $|V| \leq 2k$. □

Applying the algorithm of Theorem 4 on this kernel for subcubic graphs shows the following result.

Corollary 1. *Deciding whether a subcubic graph has a spanning tree with at least k internal vertices can be done in time $3.4854^k n^{\mathcal{O}(1)}$.*

However, we can achieve a faster parameterized running time by applying a Measure & Conquer analysis which is customized to the parameter k. We would like to put forward that our use of the technique of Measure & Conquer for a parameterized algorithm analysis goes beyond previous work as our measure is not restricted to differ from the parameter k by just a constant, a strategy exhibited by Wahlström in his thesis [24]. We first demonstrate our idea with a simple analysis.

Theorem 6. *Deciding whether a subcubic graph has a spanning tree with at least k internal vertices can be done in time $2.7321^k n^{\mathcal{O}(1)}$.*

Proof. Note that the assumption that G has no Hamiltonian path can still be made due to the $2k$-kernel of Lemma 9: the running time of the Hamiltonian path algorithm of Lemma 5 is

$1.251^{2k}n^{\mathcal{O}(1)} = 1.5651^k n^{\mathcal{O}(1)}$. The running time analysis of our algorithm relies on the following measure:

$$\kappa := \kappa(G, F, k) := k - \omega \cdot |X| - |Y| - \tilde{k},$$

where $X := \{v \in V \mid d_G(v) = 3, d_T(v) = 2\}$, $Y := \{v \in V \mid d_G(v) = d_T(v) \geq 2\}$ and $0 \leq \omega \leq 1$. Let $U := V \setminus (X \cup Y)$ and note that k in the definition of κ never changes in any recursive call of the algorithm, so neither through branching nor through the reduction rules. The variable $\tilde{k}$ counts how many times the reduction rules **ConsDeg2** and **Pending** have been applied upon reaching the current search tree node. Note that by an **ConsDeg2** the number of internal vertices goes up by one. If **Pending** is applied a vertex $v \in Y$ will be moved to U in the evolving instance G' even though v will be internal in the original instance G. This would increase κ if $\tilde{k}$ would not balance this. Note that a vertex which has already been decided to be internal, but that still has an incident edge in $E \setminus T$, contributes a weight of $1 - \omega$ to the measure. Or equivalently, such a vertex has been only counted by ω. Consider the algorithm described earlier, with the only modification that that the algorithm keeps track of κ and that the algorithm stops and answers YES whenever $\kappa \leq 0$. None of the reduction and branching rules increases κ. The explicit proof for this will skipped as it is subsumed by Lemma 11 which deals with a refined measure. We have that $0 \leq \kappa \leq k$ at any time of the execution of the algorithm. In step 4, whenever the algorithm branches on an edge $\{a, b\}$ such that $d_T(a) = 1$ (w.l.o.g., we assume that $a \in V(T)$), the measure decreases by at least ω in one branch, and by at least 1 in the other branch. To see this, it suffices to look at vertex a. Due to **Deg2**, $d_G(a) = 3$. When $\{a, b\}$ is added to F, vertex a moves from the set U to the set X. When $\{a, b\}$ is removed from G, a subsequent application of the **Deg2** rule adds the other edge incident to a to F, and thus, a moves from U to Y.

Still in step 4, let us consider the case where $d_T(a) = 2$. Then condition (b) ($d_G(b) = 2$) of step 4 must hold, due to the preference of the reduction and branching rules: condition (a) is excluded due to reduction rule **Attach**, (c) is excluded due to **Attach2** and (d) is excluded due to its condition that $d_T(a) = 1$. When $\{a, b\}$ is added to F, the other edge incident to b is also added to F by a subsequent **Deg2** rule. Thus, a moves from X to Y and b from U to Y for a measure decrease of $(1 - \omega) + 1 = 2 - \omega$. When $\{a, b\}$ is removed from G, a moves from X to Y for a measure decrease of $1 - \omega$. Thus, we have a $(2 - \omega, 1 - \omega)$-branch.

In step 5, $d_T(a) = 2$, $d_G(b) = 3$, and $d_F(b) = 0$. Vertex a moves from X to Y in each branch and b moves from U to Y in the two latter branches. In total we have a $(1 - \omega, 2 - \omega, 2 - \omega)$-branch. By setting $\omega = 0.45346$ and evaluating the branching factors, the proof follows. □

This analysis can be improved by also measuring vertices of degree 2 and vertices incident to pt-edges differently.

Theorem 7. *Deciding whether a subcubic graph has a spanning tree with at least k internal vertices can be done in time $2.1364^k n^{\mathcal{O}(1)}$.*

The proof of this theorem follows the same lines as the previous one, except that we consider a more detailed measure:

$$\kappa := \kappa(G, F, k) := k - \omega_1 \cdot |X| - |Y| - \omega_2|Z| - \omega_3|W| - \tilde{k}, \text{ where}$$

- $X := \{v \in V \mid d_G(v) = 3, d_T(v) = 2\}$ is the set of vertices of degree 3 that are incident to exactly 2 edges of T,
- $Y := \{v \in V \mid d_G(v) = d_T(v) \geq 2\}$ is the set of vertices of degree at least 2 that are incident to only edges of T,
- $W := \{v \in V \setminus (X \cup Y) \mid d_G(v) \geq 2, \exists u \in N(v) \text{ st. } d_G(u) = d_F(u) = 1\}$ is the set of vertices of degree at least 2 that have an incident pt-edge, and
- $Z := \{v \in V \setminus W \mid d_G(v) = 2, N[v] \cap (X \cup Y) = \emptyset\}$ is the set of degree 2 vertices that do not have a vertex of $X \cup Y$ in their closed neighborhood, and are not incident to a pt-edge.

We immediately set $\omega_1 := 0.5485, \omega_2 := 0.4189$ and $\omega_3 := 0.7712$. Let $U := V \setminus (X \cup Y \cup Z \cup W)$. We first have to show that the algorithm can be stopped whenever the measure drops to 0 or less.

Lemma 10. *Let $G = (V, E)$ be a connected graph, k be an integer and $F \subseteq E$ be a set of edges that can be partitioned into a tree T and a set of pending tree edges P. If none of the reduction rules applies to this instance and $\kappa(G, F, k) \leq 0$, then G has a spanning tree $T^* \supseteq F$ with at least k internal nodes.*

Proof. Since the vertices in $X \cup Y$ are internal in any spanning tree containing F, it is sufficient to show that there exists a spanning tree $T^* \supseteq F$ that has at least $\omega_2|Z| + \omega_3|W|$ more internal vertices than T.

The spanning tree T^* is constructed as follows. Greedily add a subset of edges $A \subseteq E \setminus F$ to F to obtain a spanning tree T' of G. While there exists $v \in Z$ with neighbors u_1 and u_2 such that $d_{T'}(v) = d_{T'}(u_1) = 1$ and $d_{T'}(u_2) = 3$, set $A := (A \setminus \{v, u_2\}) \cup \{u_1, v\}$. This procedure finishes in polynomial time as the number of internal vertices increases each time such a vertex is found. Call the resulting spanning tree T^*.

By connectivity of a spanning tree, we have:

Fact 1 *If $v \in W$, then v is internal in T^*.*

Note that $F \subseteq T^*$ as no vertex of Z is incident to an edge of F. By the construction of T^*, we have the following.

Fact 2 *If u, v are two adjacent vertices in G but not in T^*, such that $v \in Z$ and u, v are leaves in T^*, then v's other neighbor has T^*-degree 2.*

Let $Z_\ell \subseteq Z$ be the subset of vertices of Z that are leaves in T^* and let $Z_i := Z \setminus Z_\ell$. As $F \subseteq T^*$ and by Fact 1, all vertices of $X \cup Y \cup W \cup Z_i$ are internal in T^*. Let P denote the subset of vertices of $N(Z_\ell)$ that are internal in T^*. As P might intersect with W and for $u, v \in Z_\ell$, $N(u)$ and $N(v)$ might intersect (but $u \notin N(v)$ because of **ConsDeg2**), we assign an initial potential of 1 to vertices of P. By definition, $P \cap (X \cup Y) = \emptyset$. Thus the number of internal vertices in T^* is at least $|X| + |Y| + |Z_i| + |P \cup W|$. To finish the proof of the claim, we show that $|P \cup W| \geq \omega_2|Z_l| + \omega_3|W|$.

Decrease the potential of each vertex in $P \cap W$ by ω_3. Then, for each vertex $v \in Z_\ell$, decrease the potential of each vertex in $P_v = N(v) \cap P$ by $\omega_2/|P_v|$. We show that the potential of each vertex in P remains positive. Let $u \in P$ and $v_1 \in Z_\ell$ be a neighbor of u. Note that $d_{T^*}(v_1) = 1$. We distinguish two cases based on u's tree-degree in T^*.

$d_{T^*}(u) = 2$
- $u \in W$: Then by connectivity $\{u, v_1\} \notin T^*$ and u is incident to only one vertex out of Z_ℓ, namely v_1. Again by connectivity $h \in N(v_1) \setminus \{u\}$ is an internal vertex. Thus, the potential is $1 - \omega_3 - \omega_2/2 \geq 0$.
- $u \notin W$: u is incident to at most 2 vertices of Z_ℓ (by connectivity of T^*), its potential remains thus positive as $1 - 2\omega_2 \geq 0$.

$d_{T^*}(u) = 3$
- $u \in W$: Because $u \in W$ is incident to a pt-edge, it has one neighbor in Z_ℓ (connectivity of T^*), which has only internal neighbors (by Fact 2). The potential of u is thus $1 - \omega_3 - \omega_2/2 \geq 0$.
- $u \notin W$: u has at most two neighbors in Z_ℓ, and both of them have only inner neighbors due to Fact 2. As $1 - 2\omega_2/2 \geq 0$, u's potential remains positive. □

We also show that reducing an instance does not increase its measure.

Lemma 11. *Let (G', F', k) be an instance resulting from the exhaustive application of the reduction rules to an instance (G, F, k). Then, $\kappa(G', F', k) \leq \kappa(G, F, k)$.*

Proof. **Cycle**: If the reduction rule **Cycle** is applied to (G, F, k), then an edge in $\partial E(T)$ is removed from the graph. Then, the parameter k stays the same, and either each vertex remains in the same set among X, Y, Z, W, U, or one or two vertices move from X to Y, which we denote shortly by the status change of a vertex u: $\{X\} \to \{Y\}$ $(\omega_1 - 1)$. The value of this status change (denoted in parenthesis) is $(-1) - (-\omega_1) \leq 0$. As the value of the status change is non-positive, it does not increase the measure.

DoubleEdge: Suppose between u and v is a double edge. Then as mentioned before at most one of them belongs to T. The possible transitions for u (and v) are $\{X\} \to \{Y\}(\omega_1 - 1)$ if $d_T(u) = 2$ and $d_G(u) = 3$, $\{U\} \to \{U\}$ (0) if $d_T(u) = 1$ and $d_G(u) = 3$, $\{Z\} \to \{U\}$ (ω_2) if $d_G(u) = 2$ and $d_T(u) = 0$. Now in the last case, which has a positive status change value, we must have $d_G(v) = 3$ and $d_T(v) = 0$ or $d_G(v) = 3$ and $d_T(v) = 1$. Thus, in the first case the combined status change is $\{Z, U\} \to \{U, Z\}$ (0). In the second case immediately afterwards **Bridge** will be applied and the status change is $\{Z, U\} \to \{U, Y\}$ $(\omega_2 - 1)$.

Bridge: If **Bridge** is applied, then let $e = \{u, v\}$ with $u \in \partial_V E(T)$. Vertex u is either in U or in X, and $v \in U \cup Z \cup W$. If $v \in U$, then $v \in U$ after the application of **Bridge**, as v is not incident to an edge of T (otherwise, reduction rule **Cycle** would have applied). In this case, it is sufficient to check how the status of u can change, which is $\{U\} \to \{Y\}$ (-1) if u has degree 2, $\{U\} \to \{X\}$ $(-\omega_1)$ if $d_G(u) = 3$ and $d_T(u) = 1$, and $\{X\} \to \{Y\}$ $(\omega_1 - 1)$ if $d_G(u) = 3$ and $d_T(u) = 2$. If $v \in Z$, then v moves to U as u necessarily ends up in $X \cup Y$. The possible status changes are $\{U, Z\} \to \{Y, U\}$ $(\omega_2 - 1)$ if $d_G(u) = 2$, $\{U, Z\} \to \{X, U\}$ $(\omega_2 - \omega_1)$, if $d_G(u) = 3$ and $d_T(u) = 1$, and $\{X, Z\} \to \{Y, U\}$ $(\omega_1 + \omega_2 - 1)$ if $d_G(u) = 3$ and $d_T(u) = 2$. If $v \in W$, v ends up in X or Y, depending on whether it is incident to one or two pt-edges. The possible status changes are then $\{U, W\} \to \{Y, X\}$ $(\omega_3 - 1 - \omega_1)$, $\{U, W\} \to \{Y, Y\}$ $(\omega_3 - 2)$, $\{U, W\} \to \{X, X\}$ $(\omega_3 - 2 \cdot \omega_1)$, $\{U, W\} \to \{X, Y\}$ $(\omega_3 - \omega_1 - 1)$, $\{X, W\} \to \{Y, X\}$ $(\omega_3 - 1)$, and $\{X, W\} \to \{Y, Y\}$ $(\omega_1 + \omega_3 - 2)$.

Deg1: If **Deg1** applies, the possible status changes are $\{U\} \to \{W\}$ $(-\omega_3)$ and $\{Z\} \to \{W\}$ $(\omega_2 - \omega_3)$. Note that **Bridge** is applied before.

Pending: In **Pending**, the status change $\{W\} \to \{U\}$ (ω_3) has positive value, but the measure κ still decreases as $\tilde{k}$ also increases by 1.

ConsDeg2: Similarly, in **ConsDeg2**, a vertex in $Z \cup U$ disappears, but $\tilde{k}$ increases by 1.

Deg2: In **Deg2**, the possible status changes are $\{U\} \to \{Y\}$ (-1), $\{U, Z\} \to \{Y, U\}$ $(\omega_2 - 1)$, and $\{U, W\} \to \{Y, X\}$ $(\omega_3 - 1 - \omega_1)$.

Attach: In **Attach**, u moves from X to Y. Thus the status change for u is $\{X\} \to \{Y\}$ $(\omega_1 - 1)$. Taking into account the status of $v \in N_{V \setminus T}(u)$ another status change is $\{X, U\} \to \{Y, Z\}$ $(\omega_1 - 1 - \omega_2)$ in case $d_G(v) = 3$ and $d_F(v) = 0$. Observe that $v \in Z$ is not possible as $u \in X$.

Attach2: The only status change happens for u: $\{X\} \to \{Y\}$ $(\omega_1 - 1)$.

Special: In **Special**, the possible status changes are $\{U, Z\} \to \{X, U\}$ $(\omega_2 - \omega_1)$ and $\{X\} \to \{Y\}$ $(\omega_1 - 1)$. □

Proof. (of Theorem 7) Table 2 outlines how vertices a, b, and their neighbors move between U, X, Y, Z, and W in the branches where an edge is added to F or deleted from G in the different cases of the algorithm. For each case, the scenario giving the worst branching tuple is described.

The tight branching numbers are found for cases 4.(b) with $d_T(a) = 2$, 4.(c), 4.(d), and 5. with all of b's neighbors having degree 3. The respective branching numbers are $(2 - \omega_1 - \omega_2, 1 - \omega_1 - \omega_2 + \omega_3)$, $(2\omega_1 - \omega_3, 2)$, $(\omega_1, 1 + \omega_2)$, and $(1 - \omega_1 + \omega_2, 2 - \omega_1 + \omega_2, 2 - \omega_1 + \omega_2)$. They all yield the same basis 2.1364 of the exponential term in the running time estimate. □

4 Conclusion & Future Research

We have shown that Max Internal Spanning Tree can be solved in time $\mathcal{O}^*(2^n)$ or even better if the input graph has bounded degree. In a preliminary version of this paper we asked whether MIST can be solved in time $\mathcal{O}^*(2^n)$ and also expressed our interest in polynomial

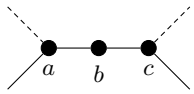
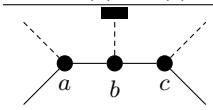
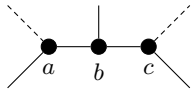
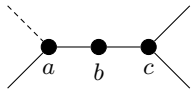
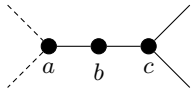
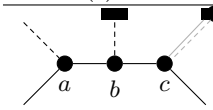
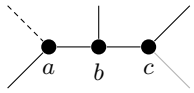
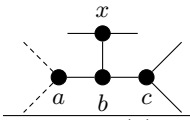
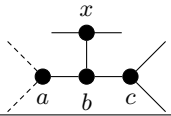
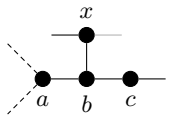
	add	delete	branching tuple
Case 4.(a), $d_G(b) = 2$			
	$a : U \to X$ $b : Z \to U$ $c : U \to Y$	$a : U \to Y$ $b : Z \to U$ $c : U \to X$	$(1 + \omega_1 - \omega_2, 1 + \omega_1 - \omega_2)$
Case 4.(a), $d_G(b) = 3$, b is incident to a pt-edge			
	$a : U \to X$ $b : W \to Y$ $c : U \to Y$	$a : U \to Y$ $b : W \to Y$ $c : U \to X$	$(2 + \omega_1 - \omega_3, 2 + \omega_1 - \omega_3)$
Case 4.(a), $d_G(b) = 3$, b is not incident to a pt-edge			
	$a : U \to X$ $b : U \to Y$ $c : U \to Y$	$a : U \to Y$ $b : U \to Z$	$(2 + \omega_1, 1 + \omega_2)$
Case 4.(b), $d_T(a) = 1$			
	$a : U \to X$ $b : Z \to Y$	$a : U \to Y$ $b : Z \to U$ $c : U \to W$	$(1 + \omega_1 - \omega_2, 1 + \omega_3 - \omega_2)$
Case 4.(b), $d_T(a) = 2$			
	$a : X \to Y$ $b : Z \to Y$	$a : X \to Y$ $b : Z \to U$ $c : U \to W$	$(2 - \omega_1 - \omega_2, 1 - \omega_1 - \omega_2 + \omega_3)$
Case 4.(c)			
	$a : U \to X$ $b : W \to X$	$a : U \to Y$ $b : W \to Y$ $c : U \to W$	$(2\omega_1 - \omega_3, 2)$
Case 4.(d)			
	$a : U \to X$	$a : U \to Y$ $b : U \to Z$	$(\omega_1, 1 + \omega_2)$
Case 5, $d_G(x) = d_G(c) = 3$ and there is a $q \in (X \cap (N(x) \cup N(c))$, w.l.o.g., $q \in N(c)$			
	$a : X \to Y$ $b : U \to Y$ (2nd branch) $q : X \to Y$	$a : X \to Y$ $b : U \to Z$	$(2 - \omega_1, 3 - 2\omega_1, 1 - \omega_1 + \omega_2)$
Case 5, $d_G(x) = d_G(c) = 3$			
	$a : X \to Y$ $b : U \to Y$ $c/x : U \to Z$ There are 3 branches; 2 of them are symmetric.	$a : X \to Y$ $b : U \to Z$	$(2 - \omega_1 + \omega_2, 2 - \omega_1 + \omega_2, 1 - \omega_1 + \omega_2)$
Case 5, $d_G(x) = 2$ or $d_G(c) = 2$, w.l.o.g., $d_G(c) = 2$			
	$a : X \to Y$ $b : U \to Y$ When $\{a, b\}$ is deleted, **ConsDeg2** additionally increases $\tilde{k}$ by 1 and removes a vertex of Z.	$a : X \to Y$ $b : U \to Z$	$(2 - \omega_1, 2 - \omega_1, 2 - \omega_1)$

Table 2. Analysis of the branching for the running time of Theorem 7

space algorithms for MIST. These questions have been settled very recently by Nederlof [17] by providing an $\mathcal{O}^*(2^n)$ polynomial-space algorithm for MIST which is based on the principle of Inclusion-Exclusion and on a new concept called "branching walks".

We focused on algorithms for MIST that work for the degree-bounded case, in particular, for subcubic graphs. The main novelty is a Measure & Conquer approach to analyze our algorithm from a parameterized perspective (parameterizing by the solution size). We are not aware of many examples where this was successfully done without cashing the obtained gain at an early stage, see M. Wahlström [24]. More examples in this direction would be interesting to see.[3] Further improvements on the running times of our algorithms pose another natural challenge.

A related problem worth investigating is the generalization to directed graphs: Find a directed tree, which consist of directed paths form the root to the leaves with as few leaves as possible. Which results can be carried over to the directed case?

Acknowledgment We thank Alexey A. Stepanov for useful discussions in the initial phase of this paper.

References

1. R. Bellman. Dynamic programming treatment of the Travelling Salesman Problem. *J. Assoc. Comput. Mach.* 9 (1962), 61–63.
2. D. Binkele-Raible. *Amortized Analysis of Exponential Time- and Parameterized Algorithms: Measure & Conquer and Reference Search Trees.* PhD thesis, Fachbereich IV, Universität Trier, Germany, 2010.
3. A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto: Fourier meets Möbius: fast subset convolution. In: *Proceedings of STOC 2007*, pages 67–74. ACM Press, 2007.
4. A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. The Travelling Salesman Problem in bounded degree graphs. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *LNCS*, pages 198–209. Springer, 2008.
5. 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 *Computing and Combinatorics, 13th Annual International Conference, COCOON 2009, Proceedings,* volume 5609 of *LNCS*, pages 37–46. Springer, 2009.
6. D. Eppstein. The Traveling Salesman problem for cubic graphs. In *J. Graph Algorithms Appl.* 11(1), pages 61–81, 2007.
7. H. Fernau, S. Gaspers, and D. Raible. Exact and parameterized algorithms for MAX INTERNAL SPANNING TREE. In *Graph-Theoretic Concepts in Computer Science, 35th International Workshop,* WG 2009, volume 5911 of *LNCS*, pages 100–111. Springer, 2010.
8. H. Fernau, S. Gaspers, D. Raible, and A. A. Stepanov. Exact Exponential Time Algorithms for Max Internal Spanning Tree. ArXiv CoRR abs/0811.1875 (2008).
9. F. V. Fomin, S. Gaspers, S. Saurabh, and S. Thomassé. A linear vertex kernel for Maximum Internal Spanning Tree. In *Algorithms and Computation, 20th International Symposium, ISAAC 2009*, volume 5878 of *LNCS*, pages 267–277. Springer, 2009.
10. F. V. Fomin, D. Lokshtanov, F. Grandoni, S. Saurabh. Sharp Separation and Applications to Exact and Parameterized Algorithms. In *Proceedings of LATIN 2010, 9th Latin American Theoretical Informatics Symposium*, volume 6034 of *LNCS*, pages 72–83. Springer, 2010.
11. M. Held, R. M. Karp. A dynamic programming approach to sequencing problems. *J. Soc. Indust. Appl. Math.* 10, pages 196–210, 1962.
12. K. Iwama and T. Nakashima. An improved exact algorithm for cubic graph TSP. In *Computing and Combinatorics, 13th Annual International Conference, COCOON 2007, Proceedings*, volume 4598 of *LNCS*, pages 108–117. Springer, 2007.

[3] Since the conference version of this paper appeared, indeed several other examples for using Measure & Conquer in connection with the analysis of parameterized algorithms have shown up, most of which are contained in the PhD Thesis of one of the authors [2].

13. R. M. Karp. Dynamic programming meets the principle of inclusion-exclusion. *Inf. Process. Lett.*, 1(2):49–51, 1982.
14. M. Knauer and J. Spoerhase. Better Approximation Algorithms for the Maximum Internal Spanning Tree Problem. In *Proceedings of WADS 2009, Workshop on Algorithms and Data Structures*, volume 5664 of *LNCS*, pages 459–470. Springer, 2009.
15. S. Kohn, A. Gottlieb, and M. Kohn. A generating function approach to the Traveling Salesman Problem. In *Proceedings of the 1977 ACM Annual Conference (ACM 1977)*, pages 294–300. Association for Computing Machinery, 1977.
16. L. M. Mays. *Water Resources Engineering*, 2nd ed. Wiley, 2010.
17. J. Nederlof. Fast polynomial-space algorithms using Möbius inversion: Improving on Steiner Tree and related problems. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5555 of *LNCS*, pages 713–725. Springer, 2009.
18. E. Prieto. *Systematic Kernelization in FPT Algorithm Design.* PhD thesis, The University of Newcastle, Australia, 2005.
19. E. Prieto and C. Sloper. Either/or: Using vertex cover structure in designing FPT-algorithms—the case of k-internal spanning tree. In *Proceedings of WADS 2003, Workshop on Algorithms and Data Structures*, volume 2748 of *LNCS*, pages 465–483. Springer, 2003.
20. E. Prieto and C. Sloper. Reducing to independent set structure – the case of k-internal spanning tree. *Nord. J. Comput.*, 12(3): 308–318, 2005.
21. G. Salamon and G. Wiener. On finding spanning trees with few leaves. *Inf. Process. Lett.*, 105(5): 164–169, 2008.
22. G. Salamon. Approximation algorithms for the maximum internal spanning tree problem. *Theor. Comput. Sci.*, 410(50): 5273–5284, 2009.
23. G. Salamon. A survey on algorithms for the maximum internal spanning tree and related problems. *Electronic Notes in Discrete Mathematics*, 36:1209–1216, 2010.
24. M. Wahlström. *Algorithms, Measures and Upper Bounds for Satisfiability and Related Problems.* PhD thesis, Department of Computer and Information Science, Linköpings universitet, Sweden, 2007.