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] Vienna Univ. of Technology, Favoritenstraße 9-11, A-1040 Wien, Austria
gaspers@kr.tuwien.ac.at
[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 $\mathcal{O}(1.8612^n)$ 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. [4] 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). The third author acknowledges partial support from the ERC, grant reference 239962. A preliminary version of this paper appeared in the proceedings of WG 2009 [15].

[4] 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. [5]

Related 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 [2], and M. Held and R. M. Karp [27]. 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.* [31] gave an algorithm based on generating functions with a running time of $\mathcal{O}(2^n n^3)$ and a space requirement of $\mathcal{O}(n^2)$ and in 1982 R. M. Karp [29] came up with an algorithm which improved storage requirements to $\mathcal{O}(n)$ and preserved this running time by an inclusion-exclusion approach. Quite recently, A. Björklund [8] devised a randomized (Monte Carlo) algorithm running in time $\mathcal{O}(1.657^n)$, using polynomial space.

D. Eppstein [13] studied TSP on cubic graphs. He achieved a running time of $\mathcal{O}(1.260^n)$ using polynomial space. K. Iwama and T. Nakashima [28] improved this to $\mathcal{O}(1.251^n)$. A. Björklund *et al.* [7] 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 $\mathcal{O}(1.8557^n)$-algorithm and for $d = 5$, an $\mathcal{O}(1.9320^n)$-algorithm. Using an inclusion-exclusion approach, J. Nederlof [36] (see also [32]) 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.* [23] recently used a *divide & conquer* approach to solve a generalization of MIST in $\mathcal{O}(2^{n+o(n)})$ 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 [40, 39] proved a $\mathcal{O}(k^3)$-vertex kernel for the problem showing $\mathcal{FPT}$-membership. The same authors [41] improved the kernel size to $\mathcal{O}(k^2)$ and F. V. Fomin *et al.* [19] to $3k$. Parameterized algorithms for MIST have been studied in [12, 19, 41]. E. Prieto and C. Sloper [41] gave the first $\mathcal{FPT}$-algorithm, with running time $2^{4k \log k} \cdot n^{\mathcal{O}(1)}$. This result was improved by N. Cohen *et al.* [12], 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 $8^k \cdot n^{\mathcal{O}(1)}$ [19] and the currently fastest algorithm for the directed version has running time $16^{k+o(k)} + n^{\mathcal{O}(1)}$ [23].

G. Salamon [45] 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 [30] 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 [44]. They introduced algorithms with approximation ratios $\frac{6}{5}$ and $\frac{3}{2}$, respectively. Further variants of our problem are discussed in surveys [38, 46], where more pointers to the literature can be found.

[5] 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, for example, 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. We refer the reader to textbooks like [34].

Finally, let us make some remarks on the main methodology of this paper, Measure & Conquer (M&C), which is nowadays a standard approach for developing and analyzing (in most cases, quite simple) exact exponential-time algorithms for computationally hard problems, in particular graph problems; see [20] and [22] for an overview. It tries to balance worse and better situations within the algorithm analysis via a potential-function analysis of the running time. For example, Minimum Dominating Set can now be solved in time $\mathcal{O}(1.5012^n)$ [37], while it was believed for quite some time that the naïve enumerative algorithm that considers 2^n cases cannot be improved. [6] The M&C approach has even proved successful to design moderately exponential time algorithms for several *non-local problems*, where an element of the solution might directly influence elements at an arbitrary distance (adding a vertex to a solution might create cycles that involve very distant vertices in a graph, for example). Examples include algorithms for Maximum Induced Forest [18], Maximum Leaf Spanning Tree [17, 21], and Full Degree Spanning Tree [25].

A M&C analysis identifies local structures that make an instance easier to solve (e.g., vertices of degree at most 2 might be easy to handle for a specific problem), and assigns a smaller measure to instances that contain many such easy local structures. Typically, weights are associated to all local structures that can occur in an instance, and the measure of an instance is the total weight of the local structures of that specific instance (e.g., the measure of a graph might depend on the number of vertices of certain degrees). Easy local structures naturally obtain a smaller weight (e.g., vertices of degree at most 2 contribute less to the total measure). The advantage obtained from a better branching on the easy local structures can be amortized over the steps that branch on them and the steps that create or lead to these favorable situations (e.g., the analysis of a branching step that decreases the degree of a vertex to 2 or less now takes that advantage into account). The analysis of the worst-case running time amounts to go through the different cases of the algorithm, and to lower bound the decrease of the measure in each branching step. This leads to a system of constraints or recurrences that depend on the individual weights. The solution of this system that minimizes the upper bound on the running time can be computed efficiently, and with a certificate of optimality, by solving a convex program [26] (see also [24]).

In the field of exact exponential time algorithms, it is usually straightforward to upper bound the measure by a function of the parameter of interest (e.g., the measure is upper bounded by the number of vertices if each vertex may contribute at most 1 to the measure). However, if the parameter is not directly related to the instance size, applying M&C becomes much less obvious. We counter this obstacle by defining measures that start out equal to the parameter k, but that we decrease by a certain value for each easy local structure of the instance. The advantages of the M&C approach are kept: the resulting measure is obviously upper bounded by k, the measure is smaller for instances with many easy local structures, and the optimal weights can still be determined efficiently by convex programming. However, this approach immediately creates several serious issues, that are often trivial if the parameter of interest is related to the size of the instance, but that need special consideration for a parameterized M&C analysis.

(1) The measure of an instance with many easy local structures might be 0 or less. In this case, a solution needs to be found in polynomial or subexponential time. When k is related to the solution, this may be achieved by considering how a solution can intersect the simple local configurations. For example, if u is a vertex of degree 2 of a graph G, then there is a spanning tree of G in which at least 2 vertices of $N[u]$ are internal. However, there may not exist a spanning tree that shares two internal vertices with the closed neighborhood of every degree-2 vertex (a spanning tree of a cycle has two adjacent leaves, for example). Moreover, this local configuration, $N[u]$, might overlap with other local configurations for which we decrease the measure. In the proof of Lemma 11 we greedily complete the partial spanning tree that has been constructed, perform some local modifications, and prove, by a potential

[6] At least, no better algorithm was known for this natural graph-theoretic problem before 2004, in contrast to the Maximum Independent Set problem (see [35, 43] for early references).

function argument, that the resulting spanning tree has at least k internal vertices if the measure of the instance was at most 0. In order to get the biggest advantage out of the M&C analysis, one needs to identify the loosest constraints on the measure which make it possible to solve the problem for instances with measure at most 0, so that we have as small a restriction as possible when it comes to decreasing the measure due to a favorable local configuration.

(2) Given an arbitrary instance, the exhaustive application of the reduction rules must not increase the measure of an instance. In our algorithm, applying the **DoubleEdge** reduction rule might in fact increase the measure temporarily, but in this case, other reduction rules are triggered, with a net increase of the measure of at most 0 (see Lemma 12).

The novelty in our approach is that our measure is not related to the instance size (as opposed to [33], for example) and is yet able to capture all easy local configurations of an instance (as opposed to Wahlström's analysis of a 3-Hitting Set algorithm [47], for example).

Our Results. We obtain the following main results:

(a) In Section 2, we present 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 $\mathcal{O}(1.8612^n)$ is presented in Section 3 to which we also refer as our main algorithm. Its space requirements are polynomial. For that algorithm, we provide a graph family proving a lower bound of $\Omega(\sqrt[4]{2}^n)$ on the running time.

(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 Measure & Conquer in a way that, to our knowledge, is much less restrictive than any previous analysis for parameterized algorithms that were based on M&C; also see the discussion above.

Notation and Definitions. 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 (disjoint union). Instead of $S \setminus \{v\}$, we sometimes write $S - v$ for an element v of S.

With one particular exception that we will explicitly describe below (where we allow double edges), 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.[7]

A *path* in a graph $G = (V, E)$ is a sequence of distinct vertices $x_0 x_1 \dots x_k$ such that $\{x_i, x_{i+1}\} \in E$, $0 \leq i \leq k-1$. A path $x_0 x_1 \dots x_k$ is a *cycle* if $\{x_0, x_k\} \in E$ and $k \geq 2$ or $k = 1$ and a double edge connects x_0 and x_1. A *connected component* in a graph is an inclusion-maximal set C of vertices with the property that there exists a path between any two vertices $x, y \in C$. A *bridge* is an edge whose deletion increases the number of connected components. A graph is *connected* if it consists of only one connected component. A graph is *acyclic* or a *forest* if it contains no cycle. A connected acyclic graph is also called a *tree*.

If $G = (V, E)$ is a graph, then we sometimes refer to its vertex set V as $V(G)$, and to its edge set as $E(G)$. Any graph G' with $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$ is called a *subgraph* of G.

[7] In the already mentioned exceptional case of a graph with double edges, subcubicity rather means that for each vertex v, there are at most three edges incident to v.

If G' happens to be a tree, we refer to it as a *subtree*. Slightly abusing notation, we will write $V(E')$ to denote the set of vertices that occur in edges of the edge set E', i.e., $V(E') = \bigcup_{e \in E'} e$. If E' is a subset of edges of $G = (V, E)$, then $G[E'] = (V(E'), E')$ is the *subgraph induced by* E'. So, we will sometimes specify subgraphs by edge sets and therefore, slightly abusing notations, view subtrees as edge sets.

In a tree T, vertices of degree 1 are called *leaves* and vertices of degree at least two are called *internal* vertices. Let $\imath(T)$ denote the number of internal vertices and $\ell(T)$ the number of leaves of T. A *d-vertex* u is a vertex with $d_T(u) = d$ 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* of a graph $G = (V, E)$ is a path on $|V|$ vertices. 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.

2 Max Internal Spanning Tree on General Graphs

To decide the existence of a Hamiltonian path, Held and Karp used a dynamic programming recurrence equation [27]. 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 [16] 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 [6]. 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 backtracking techniques, it is easy to modify the algorithm so that it indeed returns a spanning tree with this number of leaves.

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

– If $G[S]$ is connected, then

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

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

– 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 $T = T^+ - v$ 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 $T^* - v = T_1$ 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 recurrence equations 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 fewest 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 $V \setminus S$, $S_1 - v$, $S_2 - v$, $\{v\}$. Since the values of Opt^L and Opt^I are stored for every possible subset $S \subseteq V$ and every vertex $v \in V$, it follows 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 [27], the problem of solving the MIST problem in time $\mathcal{O}^*(2^n)$ is an intriguing question. This question was settled by Nederlof in [36] 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 [36] nor the approach of Fomin *et al.* [23] seem to be able to beat a running time of $\mathcal{O}^*(2^n)$ for degree-bounded graphs.

In [6], 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.

Following Björklund *et al.* [6] (see also Chapter 7 in [22]), a fast evaluation of the convolution $f * g$ can be done in time $\mathcal{O}^*(2^n)$ using the following algorithm. Here we assume that $\mathcal{U}$ is a ground set of size n and the functions f, g and h are from the subsets of $\mathcal{U}$ to the set of integers $\{-M, \ldots, M\}$ where $M \leq 2^{n^{\mathcal{O}(1)}}$.

First we define the *ranked zeta transforms* of f and g by

$$f\zeta(k, S) = \sum_{\substack{W \subseteq S \\ |W|=k}} f(W) \quad \text{and} \quad g\zeta(k, S) = \sum_{\substack{W \subseteq S \\ |W|=k}} g(W). \tag{1}$$

By using dynamic programming, these ranked zeta transforms of f and g can be computed in $\mathcal{O}^*(2^n)$-time, for all subset $S \subseteq \mathcal{U}$. We omit the details of the dynamic programming algorithm that can be found in [6, 22]. Second, the *ranked convolution* of the zeta transforms, defined as

$$(f\zeta \circledast g\zeta)(k, S) = \sum_{j=0}^{k} f\zeta(j, S) \cdot g\zeta(k - j, S), \tag{2}$$

is computed by using the values of the ranked zeta transforms. (Note that the ranked convolution is done over the parameter k.)

Finally, the obtained result is inverted via the *Möbius transform* where, given a function h, its Möbius transform $h\mu$ is defined as

$$h\mu(S) = \sum_{X \subseteq S} (-1)^{|S \setminus X|} h(X). \tag{3}$$

Again, the Möbius transform can be computed in $\mathcal{O}^*(2^n)$ by dynamic programming [6, 22]. The correctness of the approach is shown by the following formula which is proved in [6, 22]:

$$(f * g)(S) = \sum_{X \subseteq S} (-1)^{|S \setminus X|} (f\zeta \circledast g\zeta)(|S|, X). \tag{4}$$

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:

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

It is shown in [6] that the approach for computing a convolution over the integer sum-product ring can be extended to the integer min-sum semiring by, still working over the sum-product ring, but eventually scaling the functions f and g.

We are now ready to explain how the fast convolution is used to compute the recurrences of the dynamic programming approach given in Section 2.1. 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 $|X| \leq 2$ can be done in $\mathcal{O}(n^2)$ time (see Section 2.1).

Assume that $X \subseteq V$, $|X| = l \geq 3$, $x \in X$, 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.

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

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

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 [6]). 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 speed up the algorithm of the previous subsection for bounded-degree graphs. In [7], Björklund *et al.* show that the number of connected vertex sets is smaller than 2^n in graphs with bounded degree.[8]

Lemma 3 (Lemma 3 in [7]). *An n-vertex graph with maximum vertex degree Δ has at most $\beta_\Delta^n + n$ connected vertex sets with $\beta_\Delta = (2^{\Delta+1} - 1)^{1/(\Delta+1)}$.*

These connected vertex sets can then be enumerated with a polynomial-delay enumeration algorithm, where the *delay* of an enumeration algorithm is the maximum time elapsed between any two consecutive outputs, and from the start of the algorithm to the first output.

Theorem 3 ([1]). *There is a $\mathcal{O}(n+m)$-space algorithm enumerating all connected vertex sets of any input graph G with delay $O(nm)$, where n is the number of vertices and m is the number of edges of G.*

This makes it possible to combine the bound provided by Lemma 3 to speed up the algorithm described in Section 2.2 on graphs of bounded degree. The idea is to go only though the connected vertex sets of a graph, which are produced by the algorithm of Theorem 3, while applying the convolution algorithm.

Lemma 4. *To compute a spanning tree with a minimum number of leaves, the fast subset convolution algorithm of Section 2.2 need only consider subsets that are connected vertex subsets.*

Proof. Let $G = (V, E)$ be a connected graph and let $\mathcal{C}$ be the set of connected vertex subsets of G. To compute a minimum leaf spanning tree, fast subset convolution is used by our algorithm to compute the values $\mathsf{Opt}^I[S, v]$ for all $S \subseteq V$ and $v \in S$. Whenever S is not connected, the value of $\mathsf{Opt}^I[S, v]$ is (by definition) set to ∞ as $G[S]$ has no spanning tree. As recalled in Section 2.2, to compute the convolution of two functions as given by Equation (4), the fast subset convolution algorithm works in three steps (see also [6, 22]):

1. Compute (by dynamic programming) the ranked zeta transforms of f and g (see (1));
2. Compute the ranked convolution of the zeta transforms $f\zeta$ and $g\zeta$ (see (2));
3. Invert the obtained result by computing (by dynamic programming) the Möbius transform (see (3)).

The convolution formula (4) is of interest only for sets S where $S \in \mathcal{C}$, since otherwise $\mathsf{Opt}^I[S, v] = \infty$. Thus, whenever the third step of the convolution algorithm is applied, the sum of (3) is done over all subsets $X \subseteq S$ where $S \in \mathcal{C}$. We claim that only sets $X \in \mathcal{C}$ are of interest. Assume that $X \notin \mathcal{C}$ and $|X| \leq |S| - 2$. (The case $|X| = |S| - 1$ will be considered later, and the case $|X| = |S|$ implies that $X = S$ and thus X is connected.) Then the ranked convolution given by Equation (2) is done for all j from 0 up to $|S|$. When $j = |X|$, the ranked zeta transform of Equation (1) applied on f requires that $W = X$ and thus $f(W) = \infty$ whereas, at the same time, the ranked zeta transform applied on g needs to consider all subsets X of size $|S| - j = 2$. Since X is not connected, there are two vertices $a, b \in X$ such that there is no path between a and b. It follows that at some point when computing the ranked convolution of g, Equation (2) considers the set $W = \{a, b\}$ (of size $|S| - j = 2$), and thus $g\zeta(2, X)$ is equal to ∞. Consequently, the value of the ranked convolution is ∞. Suppose now that $X \notin \mathcal{C}$ and $|X| = |S| - 1$. Then

[8] As a historical aside, let us mention that such kind of estimates for degree-bounded graphs were first used in the area of exact exponential-time algorithms in [43] and then further exploited in [9].

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 of Theorem 4 according to the maximum degree Δ.

again, for $j = |S| - 1$, $f\zeta(j, X)$ is equal to ∞ since $f(W) = \infty$ for $W = X$, and at the same time $g\zeta(|S| - j, X) = g\zeta(1, X) = \infty$ by definition since $|W| = 1$. Thus, whenever a set $X \notin \mathcal{C}$ is considered in (3), the result of (2) is ∞. Assume now that $X \in \mathcal{C}$ in Equation (3). Then the ranked convolution requires to compute the ranked zeta transform. Again, in (1), only sets $W \in \mathcal{C}$ are of interest, since otherwise $f\zeta(j, X)$ is ∞.

As a consequence, both in (3) and in (1) the only sets that need to be considered are the ones corresponding to a connected vertex set of G. □

The following Theorem is a consequence of Lemmata 3 and 4 and Theorem 3.

Theorem 4. *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.

3 Subcubic Maximum Internal Spanning Tree

In this section, we focus on graphs with maximum degree at most three. This main section of the paper is structured as follows: In Subsection 3.1, we collect some useful observations. Then, in Subsection 3.2, we give some reduction rules that our algorithm will apply in polynomial time to simplify the graph instance. We also prove the correctness of these rules. Subsection 3.3 describes our algorithm, which is a branching (search tree) algorithm which exhaustively applies the already mentioned reduction rules before a recursive branch. Subsection 3.4 gives details of a Measure & Conquer analysis of our algorithm. Subsection 3.5 shows an example of a graph family where our algorithm needs exponential time. Finally, in Subsection 3.6, a parameterized algorithm analysis 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

For a spanning tree T, let t_i^T denote the number of vertices u such that $d_T(u) = i$. 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 5 ([40]). *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 5 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 6. Hamiltonian Path *can be solved in time $\mathcal{O}(1.251^n)$ on subcubic graphs.*

Proof. Let $G = (V, E)$ be a subcubic graph. First, guess (by considering all the possibilities) the start and end vertices u and v, together with the start and end edges e and f of a Hamiltonian path in G, if any exists at all. Then remove all the edges incident to u and v but e and f, add a dummy edge $\{u, v\}$, and run the algorithm of [28] to find a Hamiltonian cycle in this new graph $G_{e,f}$. If it succeeds, G clearly also has a Hamiltonian path. If G has a Hamiltonian path, then some guess will yield a graph $G_{e,f}$ with a Hamiltonian cycle. This algorithm 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 by an exchange-type argument.

Lemma 7. *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 $\imath(T') \geq \imath(T)$ and $d_{T'}(u) = d_{T'}(v) = 2$.*

Proof. By removing $\{u, v\}$, T is separated into two connected components 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 $\imath(T') \geq \imath(T)$. □

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 (i.e., a forest) which will be part of the final solution. A *pending tree edge*, or *pt-edge* for short, $\{x, v\} \in F$ is an edge with one end point x of degree one (in G) and the other end point $v \notin V(T)$, see Fig. 1(i) where $\{p, v\}$ is a pt-edge.

The following *invariant* will always be true:

(1) $G[F]$ consists of a tree T and a set P of pt-edges.
(2) $G[F]$ has $1 + |P|$ connected components.
(3) G is connected.

The invariant holds by induction. 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; see Remark 1. For item 3 of the invariant to remain true, we think of edge-induced subgraphs when removing edges from the graph. Thus, isolated vertices are implicitly removed from an instance.

Next we present a sequence of reduction rules (see also Fig. 1). 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 T.

DoubleEdge: If there is a double edge, then delete one of the two edges that is not in F.

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

Deg1: If there is a vertex $u \in V \setminus V(F)$ with $d_G(u) = 1$, then add its incident edge to P.

Pending: If there is a vertex v with $d_G(v) \geq 2$ 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.

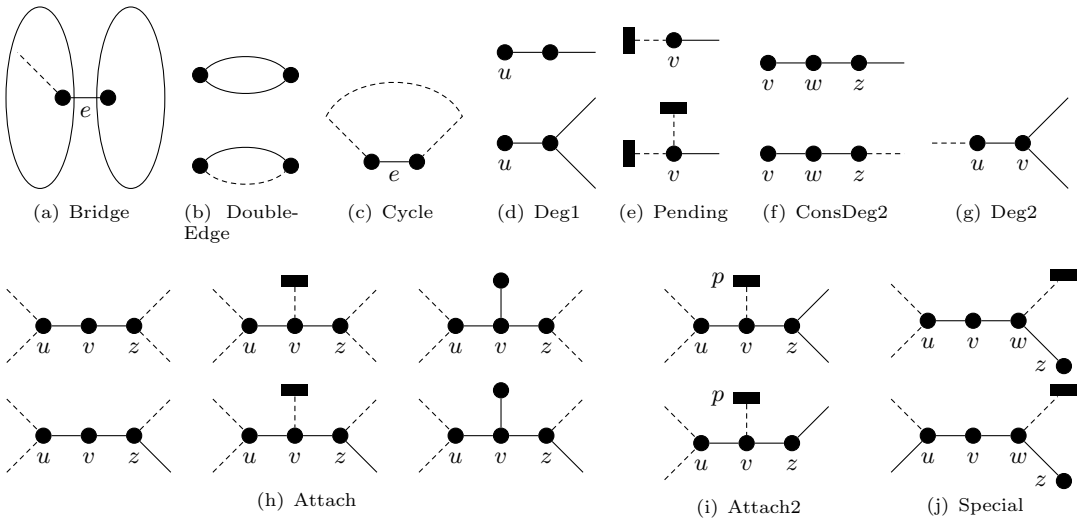


Fig. 1. Local configurations to which the different reduction rules apply. The following drawing conventions apply to all figures of this article. Dashed edges are in F, solid edges are in $E \setminus F$. Edges incident to oblongs are pt-edges and the oblong represents the vertex of degree one in this case.

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 $\{u, v\}$.

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 $\{u, v\}$.

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 T.

Remark 1. 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.

Example 1. Let us illustrate the effect of the reduction rules with a small example. Assume we are dealing with an instance (G, F) where $G = (V, E)$ is a simple graph and $F = T \uplus P$, such that the vertex sequence $v_0v_1v_2v_3$ defines a path and $d(v_0) = 3$, $d(v_1) = d(v_2) = 2$, and $d(v_3) = 1$ (see Figure 2). Moreover, assume that $\{v_0, v_1, v_2, v_3\} \cap V(T) = \emptyset$. Let us concentrate on this part of the graph.
Now, check rule **Bridge**. Clearly, G contains bridges, namely $\{v_i, v_{i+1}\}$ for $0 \leq i < 3$. However, since $\{v_0, v_1, v_2, v_3\} \cap V(T) = \emptyset$, none of these edges is in $\partial E(T)$. Hence, **Bridge** does not apply to this part of the graph.
By assumption, there are no double edges.
If **Cycle** applied, it would not delete any of the edges $\{v_i, v_{i+1}\}$ for $0 \leq i < 3$. Moreover, $V(T)$ would not be affected, so that the situation we are considering would prevail.
However, **Deg1** applies and puts $\{v_2, v_3\}$ into P.
We would then restart testing the applicability of each rule. Now, the first applicable rule is **Pending**: v_2 is incident to only one edge from $E \setminus P$. This rule removes the edge $\{v_2, v_3\}$.
The next search of an applicable rule finds **Deg1**, since v_2 is now of degree one, and puts $\{v_1, v_2\}$

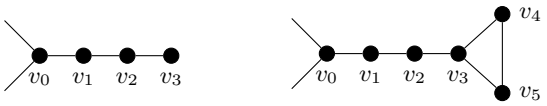


Fig. 2. The graphs from Example 1 illustrating the reduction rules.

into P; this edge is also deleted by **Pending**. Finally, the edge $\{v_0, v_1\}$ is put into P by **Deg1**. Since $d(v_0) = 3$, **Pending** does not trigger.

Assume now that the original graph G is modified by adding a triangle and identifying v_3 with one of its vertices, i.e., we add two extra vertices v_4 and v_5 and the edges $\{v_3, v_4\}$, $\{v_4, v_5\}$, and $\{v_3, v_5\}$. Now, the first rule that can be applied is **ConsDeg2**; actually there are two places to be considered. The algorithm might first discover that v_1 and v_2 are two consecutive vertices of degree two. Hence, it adds the edge $\{v_0, v_2\}$ and deletes the edges $\{v_0, v_1\}$ and $\{v_1, v_2\}$. Similarly, v_4 and v_5 are two consecutive vertices of degree two. Hence, an edge $\{v_3, v_5\}$ is added and $\{v_3, v_4\}$ and $\{v_4, v_5\}$ are deleted. This creates a double edge that is turned into a single edge by **DoubleEdge**. So, we are left with a graph that is completely isomorphic to the graph G that we already considered above, since $v_0v_2v_3v_5$ forms a path.

Theorem 5. *The reduction rules stated above are correct and can be exhaustively carried out in polynomial time.*

Proof. Notice that an instance of our algorithm is a subcubic graph $G = (V, E)$ together with an edge set $F = T \uplus P$ satisfying the stated invariant. We first argue for the correctness of the rules. Since each reduction rule can be viewed as a transformation $G \mapsto G'$ and $F \mapsto F'$, this means the following things:

- If T'_o is a spanning tree of G' with a maximum number of internal vertices subject to the condition that T'_o extends F', i.e., $T'_o \supseteq F'$, then we can derive from T'_o a spanning tree $T_o \supseteq F$ of G with a maximum number of internal vertices.
- The stated invariant ist maintained.
- If G is subcubic, then G' is subcubic.
- Remark 1 is ensured.

Regarding connectivity of the graph (item 3 of the invariant), notice that the graph G' resulting from G is always described via edge modifications. Hence, we actually describe an edge set E' and then define $G' = (V(E'), E')$; i.e., isolated vertices are removed.

Consider a subcubic graph $G = (V, E)$ together with an edge set $F = T \uplus P$ satisfying the stated invariant. The invariant implies that in the case that G contains a double edge, at most one of the involved edges can be in F. As **DoubleEdge** deletes one of the two edges that is not in F, all later reduction rules consider only simple graphs. Also notice that the preservation of subcubicity becomes only an issue if edges are added to G. Let $T_o \supset F$ be a spanning tree of G with a maximum number of internal vertices.

Bridge Consider a bridge $e \in \partial E(T)$. Since T_o is connected, $e \in T_o$ is enforced. The invariant is maintained, since e is added to T. Notice that e is not in a double edge.

DoubleEdge Since T_o is acyclic, at most one of the two edges between, say, u and v, can be in T_o.

Cycle Since T_o is acyclic, no edge $e \in E$ such that $T \cup \{e\}$ has a cycle can belong to T_o.

Deg1 Let $u \in V \setminus V(F)$ with $d(u) = 1$. Since T_o is spanning, it contains an edge incident to u. Since $d(u) = 1$, the unique edge e incident to u is in T_o. Hence, we can safely add e to F,

i.e., $F' = F \cup \{e\}$. Because $u \in V \setminus V(F)$, $F' \neq F$, and since we would have added e to T already if $e \in \partial E(T)$ due to the **Bridge** rule, we add e in fact to P, hence mantaining the invariant.

Pending Consider the only non-pt-edge $e = \{u, v\}$ incident to v. Obviously, e is a bridge and needs to be in any spanning tree of G. Moreover, all pt-edges incident to v are in F due to a previous application of the **Deg1** rule. Being incident to pt-edges means $v \notin V(T)$. Yet, v must be reached in T_o from u. Removing all pt-edges incident to v will render v a vertex of degree one in G'. Hence, the **Deg1** rule will trigger when applying the reduction rules again. This ensures that v will be reached via u, so that in the neighborhood of v, all pt-edges can be safely removed (from P). Observe that the invariant is maintained.

ConsDeg2 Let G' be the graph obtained from G by one application of **ConsDeg2**. We assume, without loss of generality, that $\{w, z\} \in T_o$. If $\{w, z\} \notin T_o$ then $\{v, w\} \in T_o$ and we can simply exchange the two edges giving a solution $\tilde{T}_o$ with $\{w, z\} \in \tilde{T}_o$ and $t_2^{T_o} \leq t_2^{\tilde{T}_o}$. (As in Subsec. 3.1, t_i^S denotes the number of vertices u such that $d_S(u) = i$ for some tree S.) By contracting the edge $\{w, z\}$, we receive a solution T'_o such that $\imath(T'_o) = \imath(T_o) - 1$.
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 $\imath(T_o) = \imath(T'_o) + 1$. If $\{v, z\} \notin T'_o$, then by connectivity $d_{T'_o}(z) = 1$. Let $T_o = T'_o \cup \{\{v, w\}\}$; then $\imath(T_o) = \imath(T'_o) + 1$.
Since the vertex w is a neighbor of v and z in G and is missing in G', adding the edge $\{v, z\}$ to G' preserves subcubicity. This edge addition might create a double-edge (in case v and z were neighbors in G), but the new edge is not in F, so that Remark 1 is maintained by induction.

Deg2 Consider an edge $\{u, v\} \in \partial E(T)$ such that $u \in V(T)$ and $d_G(u) = 2$. Since the preceding reduction rules do not apply, we have $d_G(v) = 3$. Assume $\{u, v\} \notin T_o$. There is exactly one edge incident to v, say $e = \{v, z\}$, $z \neq u$, that is contained in the unique cycle in $T_o \cup \{\{u, v\}\}$. Clearly, being part of a cycle means that e is not pending. Define another spanning tree $\tilde{T}_o \supset F$ by setting $\tilde{T}_o = (T_o \cup \{\{u, v\}\}) \setminus \{v, z\}$. Since $\imath(T_o) \leq \imath(\tilde{T}_o)$, $\tilde{T}_o$ is also optimal. So, without loss of generality, we can assume that $\{u, v\} \in T_o$.

Attach If $\{u, v\} \in T_o$ then $\{v, z\} \notin T_o$ due to the acyclicity of T_o and as T is connected. Then by exchanging $\{u, v\}$ and $\{v, z\}$ we obtain a solution $\tilde{T}_o$ 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. First, suppose $\{v, z\} \in T_o$. Due to Lemma 7, there is also an optimal solution $\tilde{T}_o \supset F$ with $\{u, v\} \notin \tilde{T}_o$. Second, assume $\{v, z\} \notin T_o$. Then $\tilde{T}_o = (T_o - \{u, v\}) \cup \{\{v, z\}\}$ 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. Clearly, $\{w, z\}$ and $\{v, w\}$ are on the cycle that is contained in $G[T_o \cup \{\{u, v\}\}]$. Hence, $\tilde{T}_o := (T_o - \{v, w\}) \cup \{\{u, v\}\}$ is also a spanning tree that extends F. In $\tilde{T}_o$, w is a 2-vertex and hence $\tilde{T}_o$ is also optimal.

It is clear that the applicability of each rule can be tested in polynomial time when using appropriate data structures. If triggered, the required operations will also use polynomial time. Finally, since the reduction rules always make the instance smaller in some sense (either by reducing the number of edges or by by reducing the number of edges not in F), the number of successive executions of reduction operations is also limited by a polynomial. □

Remark 2. It is interesting to observe how the optimum value $\imath(T_o)$ for the instance (G, F) relates to the value $\imath(T'_o)$ for the instance (G', F') obtained by applying any of the reduction rules once. Reconsidering the previous proof, one easily derives that only two situations occur: $\imath(T_o) = \imath(T'_o)$ for all rules apart from **Pending** and from **ConsDeg2** where we have $\imath(T_o) = \imath(T'_o) + 1$.

An instance to which none of the reduction rules applies is called *reduced*.

Lemma 8. *A reduced instance (G, F) with $F = T \uplus P$ has the following properties: (1) for all $v \in \partial_V E(T)$, $d_G(v) = 3$ and (2) for all $u \in V$, $d_P(u) \leq 1$.*

Proof. Assume that for some $v \in \partial_V E(T)$, $d_G(v) = 2$; then **ConsDeg2** or **Deg2** would apply, contradicting that (G, F) is reduced. If there is a $v \in \partial_V E(T)$ with $d_G(v) = 1$, then **Bridge** or **Deg1** would apply, again contradicting that (G, F) is reduced. The second property follows from the invariant. □

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.

In the initial phase, we select any vertex v and go over all subtrees of G on 3 vertices containing v. These are our initial choices for T. As every vertex has degree at most 3, there are at most 3 trees on 3 vertices with both edges incident to v, and there are at most $3 \cdot 2$ trees on 3 vertices with only one edge incident to v. For each such choice of T, we grow T with our recursive algorithm, which proceeds 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) $d_T(a) = 1$.

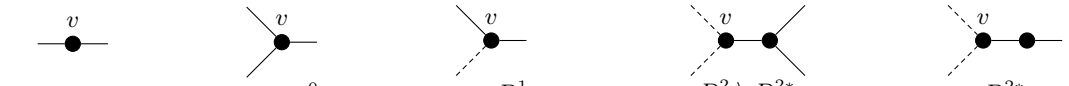
   Recursively solve the two instances where $\{a, b\}$ is added to F or removed from G respectively, and return the spanning tree with most internal vertices.
5. Otherwise, select $\{a, b\} \in \partial E(T)$ with $a \in V(T)$. Let c, x be the other two neighbors of b. Recursively solve three instances 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 6. MIST *can be solved in time $\mathcal{O}(1.8612^n)$ on subcubic graphs.*



$v \in D_2$ $\omega_2 = 0.2981$ | $v \in D_3^0$ weight 1 | $v \in D_3^1$ $\omega_3^1 = 0.6617$ | $v \in D_3^2 \setminus D_2^{2*}$ $\omega_3^2 = 0.31295$ | $v \in D_2^{2*}$ $\omega_3^{2*} = 0.4794$

Fig. 3. How much a vertex v contributes to the measure μ.

Let

- $D_2 := \{v \in V \mid d_G(v) = 2, d_F(v) = 0\}$ denote the vertices of degree 2 with no incident edge in F,
- $D_3^\ell := \{v \in V \mid d_G(v) = 3, d_F(v) = \ell\}$ denote the vertices of degree 3 with ℓ incident edges in F, and
- $D_3^{2*} := \{v \in D_3^2 \mid N_G(v) \setminus N_F(v) = \{u\}$ and $d_G(u) = 2\}$ denote the vertices of degree 3 with 2 incident edges in F and whose neighbor in $V \setminus V(T)$ has degree 2.

Let $V_{\text{out}} = V \setminus (D_2 \cup D_3^0 \cup D_3^1 \cup D_3^2)$ denote the vertices that do not contribute to the measure; their weights already dropped to zero. Then the measure we use for our running time bound is

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

with the weights $\omega_2 = 0.2981$, $\omega_3^1 = 0.6617$, $\omega_3^2 = 0.31295$ and $\omega_3^{2*} = 0.4794$. Figure 3 is meant to serve as a reference to quickly determine the weight of a vertex.

In the very beginning, $F = \emptyset$, so that $V = D_2 \uplus D_3^0$. Hence, $\mu(G, F) \leq n$. Our algorithm makes progress by growing F and by reducing the degree of vertices. This is reflected by reducing the weights of vertices within the neighborhood of F. The special role of vertices of degree two is mirrored in the sets D_2 and D_3^{2*}.

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 9. *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. Notice 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, since before the deletion, $u \in D_3^\ell$ with $\ell \geq 1$ or $u \in V_{\text{out}}$ and afterwards, $u \in V_{\text{out}}$. 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 6) 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, Lemma 8 applies.

By lower bounding the measure decrease in each recursive call, we obtain branching vectors $(a_1, a_2, \ldots, a_\ell)$, where a_i is the lower bound of the decrease of the measure in the i^{th} recursive call. If in each case the algorithm considers, $\sum_{i=1}^{\ell} c^{-a_i} \leq 1$ for a positive real number c, we obtain that the algorithm has running time $\mathcal{O}^*(c^n)$ by well-known methods described in [24], for example, because the measure is upper bounded by n and the reduction rules do not increase the measure.

We consider Cases 4 and 5 and lower bound the decrease of the measure in each branch (recursive call).

4.(a) Obviously, $\{a,b\}, \{b,c\} \in E \setminus T$, and there is a vertex d such that $\{c,d\} \in T$; see Fig. 4(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 $\{b,c\}$. The measure decreases by $\omega_2 + \omega_3^1$, as b moves from D_2 to V_{out} and c moves from D_3^1 to V_{out}. Also, a is removed from D_3^1 and added to D_3^2 which amounts to a decrease of the measure of at least $\tilde{\Delta}_3^1$. In total, the measure decreases by at least $\omega_2 + \omega_3^1 + \tilde{\Delta}_3^1$ when $\{a,b\}$ is added to F. When $\{a,b\}$ is deleted, $\{b,c\}$ is added to T (by reduction rule **Bridge**). The vertex a now has degree 2 and one incident edge in F (this vertex will trigger reduction rule **Deg2**, but we do not necessarily need to take into account the effect of this reduction rule, as reduction rules do not increase μ by Lemma 9), so a moves from D_3^1 to V_{out} for a measure decrease of ω_3^1. The vertex b moves from D_2 to V_{out} for a measure decrease of ω_2 and c moves from D_3^1 to D_3^2 for a measure decrease of $\tilde{\Delta}_3^1$. Thus, the measure decreases by at least $\omega_2 + \omega_3^1 + \tilde{\Delta}_3^1$ in the second branch, 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 $2\omega_3^1$ (from b and c, as **Cycle** deletes $\{b,c\}$). By deleting $\{a,b\}$, μ decreases by $2\omega_3^1$ (from a and b) and by $\tilde{\Delta}_3^1$ (from c, as **Bridge** adds $\{b,c\}$ to F). 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 $\omega_3^1 + \tilde{\Delta}_3^1 + 1$, namely ω_3^1 from c as **Cycle** removes $\{b,c\}$, $\tilde{\Delta}_3^1$ from a, and 1 from b. In the second branch, we get a measure decrease of ω_3^1 from a and Δ_2 from b. This results in a $(\omega_3^1 + \tilde{\Delta}_3^1 + 1, \omega_3^1 + \Delta_2)$-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 $d_T(c) = 0$. Moreover, $d_G(c) \geq 3$ due to **Pending** and **ConsDeg2**, and $d_P(c) = 0$ due to **Special**. See Fig. 4(b). We consider two subcases.

- $d_T(a) = 1$. When $\{a,b\}$ is added to F, then $\{b,c\}$ is also added by **Deg2**. The measure decreases by 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 by **Deg1**. There is $\{a,z\} \in E \setminus T$ with $z \neq b$, which is subject to a **Deg2** reduction rule. The measure decreases by 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 $(\tilde{\Delta}_3^1 + \omega_2 + \Delta_3^0, \omega_3^1 + \omega_2 + \Delta_3^0 + \min\{\omega_2, \tilde{\Delta}_m^1\})$-branch.
- $d_T(a) = 2$. In both branches, the measure decreases by ω_3^{2*} from a, by ω_2 from b, and Δ_3^0 from c. This is a $(\omega_3^{2*} + \omega_2 + \Delta_3^0, \omega_3^{2*} + \omega_2 + \Delta_3^0)$-branch.

4.(c) In this case, $d_G(b) = 3$ and there is one pt-edge attached to b, see Fig. 4(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 $\{a,z\} \in E \setminus T$. We have that $d_G(z) = 3$, as Case 4.(b) would have applied otherwise. We distinguish between the cases depending on the degree of c, the other neighbor of b, and whether c is incident to a pt-edge or not.

$d_G(c) = 2$. We distinguish whether z has an incident pt-edge. If $d_P(z) = 1$, **Attach2** is triggered when $\{a, b\}$ is added to F. This removes the edge $\{a, z\}$. The measure decrease is ω_3^1 from a, Δ_3^{1*} from b, and ω_3^1 from z. When $\{a, b\}$ is removed from G, **Deg2** adds $\{a, z\}$ to F and **Bridge** adds c's incident edges to F. The measure decrease is ω_3^1 from a, ω_3^1 from b, ω_2 from c, $\tilde{\Delta}_3^1$ from z, and $\tilde{\Delta}_m^2$ from c's other neighbor. This gives the branching vector $(2\omega_3^1 + \Delta_3^{1*}, 2\omega_3^1 + \omega_2 + \tilde{\Delta}_3^1 + \tilde{\Delta}_m^2)$.
If $d_P(z) = 0$, the analysis is similar, except that **Attach2** is not triggered in the first branch and the measure decrease incurred by z is Δ_3^0 in the second branch. This gives a $(\Delta_3^1 + \Delta_3^{1*}, 2\omega_3^1 + \omega_2 + \Delta_3^0 + \tilde{\Delta}_m^2)$-branch.

$d_G(c) = 3$ and $d_P(c) = 0$. Adding $\{a, b\}$ to F decreases the measure by $2\Delta_3^1$ (as z and c have degree 3). Deleting $\{a, b\}$, we observe a measure decrease by of $2\omega_3^1$ from a and b. As $\{a, z\}$ is added to F by **Deg2**, $\mu(G)$ decreases additionally by at least $\tilde{\Delta}_m^1$ as the state of z changes. Due to **Pending** and **Deg1**, $\{b, c\}$ is added to F and we get a measure decrease of Δ_3^0 from c. This gives a $(2\Delta_3^1, 2\omega_3^1 + \tilde{\Delta}_m^1 + \Delta_3^0)$-branch.

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

5. See Fig. 4(e). The algorithm branches in the following way:
1) Delete $\{a, b\}$ from G,
2) add $\{a, b\}, \{b, c\}$ to F, and delete $\{b, x\}$ from G, and
3) add $\{a, b\}, \{b, x\}$ to F, and delete $\{b, c\}$ from G.

Observe that these cases are sufficient to find an optimal solution. Due to **Deg2**, we can disregard the case when b is a leaf. Due to Lemma 7, we also disregard the case when b is a 3-vertex as $\{a, b\}$ is not a bridge. Thus this branching strategy finds at least one optimal solution.

The measure decrease in the first branch is at least $\omega_3^2 + \Delta_2$ from a and b. We get an additional amount of ω_2 if $d(x) = 2$ or $d(c) = 2$ from **ConsDeg2**. In the second branch, we distinguish between three situations for $h \in \{c, x\}$. These are
$\alpha)$ $d_G(h) = 2$,
$\beta)$ $d_G(h) = 3$, $d_P(h) = 0$, and
$\gamma)$ $d_G(h) = 3$, $d_P(h) = 1$.
We first get a measure decrease of $\omega_3^2 + 1$ from a and b. Deleting $\{b, x\}$ incurs a measure decrease in the three situations by $\alpha)$ $\omega_2 + \tilde{\Delta}_m^2$ by **Deg1** and **Pending**, $\beta)$ Δ_2, and $\gamma)$ $\omega_3^1 + \tilde{\Delta}_m^2$ by **Pending** and **Deg1**. Next we examine the amount by which μ decreases by adding $\{b, c\}$ to F. The measure decrease in the different situations is: $\alpha)$ $\omega_2 + \tilde{\Delta}_m^2$ by **Deg2**, $\beta)$ Δ_3^0, and $\gamma)$ $\tilde{\Delta}_3^1$. The analysis of the third branch is symmetric. For $h \in \{c, x\}$ and $\sigma \in \{\alpha, \beta, \gamma\}$ let 1_σ^h be the indicator function whose value is one if we have situation σ at vertex h. Otherwise, its value is zero. Now, the branching vector can be stated the following way:

$$\big(\omega_3^2 + \Delta_2 + (1_\alpha^x + 1_\alpha^c) \cdot \omega_2,$$
$$\omega_3^2 + 1 + 1_\alpha^x \cdot (\omega_2 + \tilde{\Delta}_m^2) + 1_\beta^x \cdot \Delta_2 + 1_\gamma^x \cdot (\omega_3^1 + \tilde{\Delta}_m^2) + 1_\alpha^c \cdot (\omega_2 + \tilde{\Delta}_m^2) + 1_\beta^c \cdot \Delta_3^0 + 1_\gamma^c \cdot \tilde{\Delta}_3^1,$$
$$\omega_3^2 + 1 + 1_\alpha^c \cdot (\omega_2 + \tilde{\Delta}_m^2) + 1_\beta^c \cdot \Delta_2 + 1_\gamma^c \cdot (\omega_3^1 + \tilde{\Delta}_m^2) + 1_\alpha^x \cdot (\omega_2 + \tilde{\Delta}_m^2) + 1_\beta^x \cdot \Delta_3^0 + 1_\gamma^x \cdot \tilde{\Delta}_3^1\big)$$

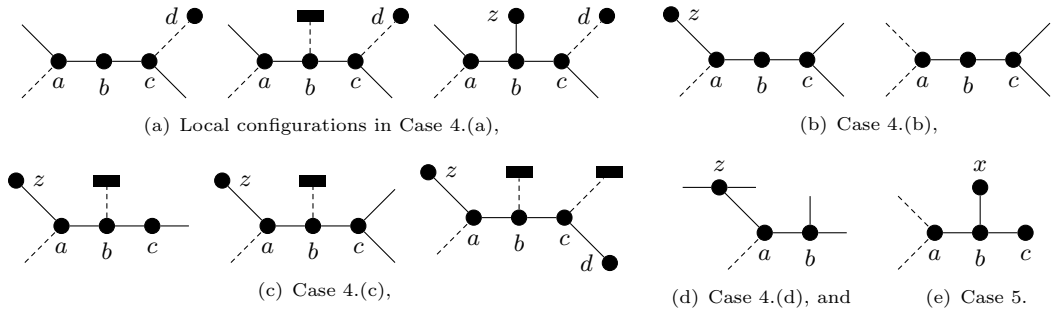


Fig. 4. Local configurations corresponding to the different cases of the algorithm where a branching occurs.

The amount of $(1^x_\alpha + 1^c_\alpha) \cdot \omega_2$ in the first branch comes from the applications of **ConsDeg2** if c and x have degree 2.

After checking that $\sum_{i=1}^{\ell} 1.8612^{-a_i} \leq 1$ for every branching vector $(a_1, a_2, \ldots, a_\ell)$, we conclude that MIST can be solved in time $\mathcal{O}^*(1.8612^n)$ on subcubic graphs. □

We conclude this analysis with mentioning the tight cases for the branching: Case 4.(b) with $d_T(a) = 1$ and $d(z) = 3$, Case 4.(b) with $d_T(a) = 2$, Case 4.(d), Case 5 with $1^x_\beta = 1$ and $1^c_\gamma = 1$, Case 5 with $1^x_\gamma = 1$ and $1^c_\beta = 1$, and Case 5 with $1^x_\gamma = 1$ and $1^c_\gamma = 1$. As there are quite some tight cases, it seems to be hard to improve on the running time upper bound by a more detailed analysis of one or two cases.

3.5 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 $\Omega(c^n)$ steps, where $c = \sqrt[4]{2} > 1.1892$ and n is the number of vertices in the input graph.

Theorem 7. *There is an infinite family of graphs for which our algorithm for* MIST *needs* $\Omega(\sqrt[4]{2}^n)$ *time.*

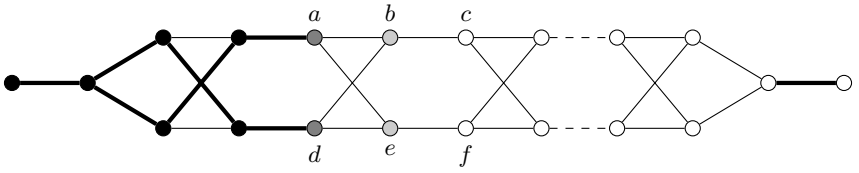


Fig. 5. Graph used to lower bound the running time of our MIST algorithm.

Proof. Instead of giving a formal description of the graph family, consider Fig. 5.

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

$\{\{d,e\},\{b,c\},\{e,f\}\}$ 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. □

3.6 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 7 carries over, but the according lower bound 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 [19]. This can be easily improved to $2k$ for subcubic graphs.

Lemma 10. *MIST on subcubic graphs has a kernel with at most $2k$ vertices.*

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 6 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 $3.4641^k n^{\mathcal{O}(1)}$.*

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 [47]. We first demonstrate our idea with a simple analysis. In this analysis, we subtract one from the measure when all incident edges of a vertex u of degree at least two are in the partial spanning tree: u becomes internal and the algorithm will not branch on u any more. Moreover, we subtract ω, a constant smaller than one, from the measure when 2 incident edges of a vertex v of degree 3 are in the partial spanning tree: v is internal, but the algorithm still needs to branch on v.

Theorem 8. *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 10: the running time of the Hamiltonian path algorithm of Lemma 6 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:

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

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_{simple} 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 node of the search tree. Note that an application of **ConsDeg2** increases the number of internal vertices by one. If **Pending** is applied, a vertex $v \in Y$ is moved to U in the evolving instance G' even though v is internal in the original instance G. This would increase κ_{simple} 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 the algorithm keeps track of κ_{simple} and that the algorithm stops and answers YES whenever $\kappa_{\text{simple}} \leq 0$. None of the reduction and branching rules increases κ_{simple}. The explicit proof for this will skipped as it is subsumed by Lemma 12 which deals with a refined measure. We have that $0 \leq \kappa_{\text{simple}} \leq k$ at any time of the execution of the algorithm. In Case 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 Case 4, let us consider the case where $d_T(a) = 2$. Then condition (b) ($d_G(b) = 2$) of Case 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 Case 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. A vertex of degree at least two that is incident to a pt-edge is necessarily internal in any spanning tree, and for a vertex u of degree 2 at least one vertex of $N[u]$ is internal in any spanning tree; we analyze in more details how the internal vertices of spanning trees intersect these local structures in the proof of Lemma 11.

Theorem 9. *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 show that the algorithm can be stopped whenever the measure drops to 0 or less.

Lemma 11. *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 $k - \tilde{k}$ internal nodes.*

Proof. As each vertex of $X \cup Y$ is incident to at least two edges of F, the vertices in $X \cup Y$ are internal in every spanning tree containing F. We will show that there exists a spanning tree $T^* \supseteq F$ that has at least $\omega_2|Z| + \omega_3|W|$ more internal vertices than F. Thus, T^* has at least $\omega_2|Z| + \omega_3|W| + |Y| + |X| \geq k - \tilde{k}$ internal vertices.

The spanning tree T^* is constructed as follows. Greedily add some 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 $T' := (T' - \{v, u_2\}) \cup \{\{u_1, v\}\}$. 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 Q denote the subset of vertices of $N(Z_\ell)$ that are internal in T^*. As Q 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 Q. By definition, $Q \cap (X \cup Y) = \emptyset$. Thus the number of internal vertices in T^* is at least $|X| + |Y| + |Z_i| + |Q \cup W|$. To finish the proof of the claim, we show that $|Q \cup W| \geq \omega_2|Z_l| + \omega_3|W|$.

Decrease the potential of each vertex in $Q \cap W$ by ω_3. Then, for each vertex $v \in Z_\ell$, decrease the potential of each vertex in $Q_v = N(v) \cap Q$ by $\omega_2/|Q_v|$. We show that the potential of each vertex in Q remains positive. Let $u \in Q$ 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 $h \in N(v_1) - u$ 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 (by the 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. □

Now, a spanning tree with at least k internal nodes of the original graph can be obtained by reversing the $\tilde{k}$ **ConsDeg2** and **Pending** operations.

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

Lemma 12. *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 parentheses) 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 **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 9) 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 vectors 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 faster if the input graph has bounded degree. In a preliminary version of this paper we asked

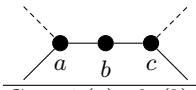
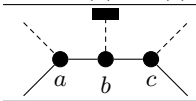
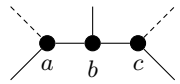
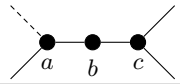
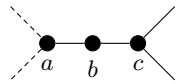
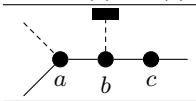
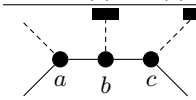
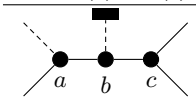
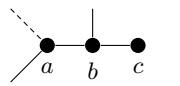
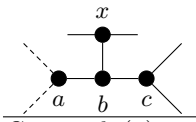
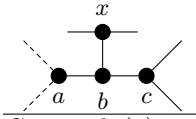
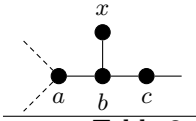
	add	delete	branching tuple
Case 4.(a), $d_G(b) = 2$			
	$a : U \to X$ $b : Z \to U$ $c : U \to Y$	$a : U \to Y$ $b : Z \to U$ $c : U \to X$	$(1 + \omega_1 - \omega_2, 1 + \omega_1 - \omega_2)$
Case 4.(a), $d_G(b) = 3$, b is incident to a pt-edge			
	$a : U \to X$ $b : W \to Y$ $c : U \to Y$	$a : U \to Y$ $b : W \to Y$ $c : U \to X$	$(2 + \omega_1 - \omega_3, 2 + \omega_1 - \omega_3)$
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), $d_G(c) = 2$			
	$a : U \to X$ $b : W \to X$	$a : U \to Y$ $b : W \to Y$ $c : U \to Y$	$(2\omega_1 - \omega_3, 3 - \omega_3)$
Case 4.(c), $d_G(c) = 3$ and $d_P(c) = 1$			
	$a : U \to X$ $b : W \to X$	$a : U \to Y$ $b : W \to Y$ $c : U \to Y$	$(2\omega_1 - \omega_3, 3 - \omega_3)$
Case 4.(c), $d_G(c) = 3$ and $d_P(c) = 0$			
	$a : U \to X$ $b : W \to X$	$a : U \to Y$ $b : W \to Y$ $c : U \to W$	$(2\omega_1 - \omega_3, 2)$
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 9

whether MIST can be solved in time $\mathcal{O}^*(2^n)$ and also expressed our interest in polynomial space algorithms for MIST. These questions were settled by Nederlof [36] 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 (parameterized by the number of internal vertices). We are not aware of many examples where this was successfully done without cashing the obtained gain at an early stage, see M. Wahlström [47]. More examples in this direction would be interesting to see.[9] Further improvements on the running times of our algorithms pose another natural challenge.

A related problem worth investigating is the analogous question for directed graphs: Find a directed (spanning) tree (usually called a (spanning) arborescence) which consist of directed paths from the root to the leaves with as many internal vertices as possible. Which results can be carried over to the directed case?

Finally, the celebrated paper of A. Björklund [8] raises several natural questions for future research:

- Is there a Monte Carlo algorithm that allows to yield further improved running time bounds for algorithms for MIST on general graphs?
- A second result of A. Björklund in the same paper [8] was a Monte Carlo algorithm for Hamiltonian Path that runs in time $2^{n-i} \cdot n^{\mathcal{O}(1)}$, where i is the size of some (available) independent set in the graph. Using Gallai's identity and some parameterized algorithm for vertex cover as described, for instance, in [10, 11], as a preprocessing, this implies a Monte Carlo algorithm for Hamiltonian Path that runs in time $2^{vc} \cdot n^{\mathcal{O}(1)}$, where vc is the vertex cover number of the input graph. This raises two further questions in the spirit of "parameter ecology" that became popular within parameterized algorithmics [14] in recent years:
  - Is there a deterministic algorithm solving Hamiltonian Path in time $c^{vc} \cdot n^{\mathcal{O}(1)}$ for a constant c not much larger than 2?
  - Is there a Monte Carlo or even a deterministic algorithm solving MIST in time $c^{vc} \cdot n^{\mathcal{O}(1)}$ for a constant c not much larger than 2?
- Can we work out the previous ideas more sucessfully for degree-bounded cases?

Acknowledgment We thank Alexey A. Stepanov for useful discussions in the initial phase of this paper and also the referees for their careful reading of the original submission.

References

1. D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics* 65(1-3):21–46, 1996.
2. R. Bellman. Dynamic programming treatment of the Travelling Salesman Problem. *Journal of the Association for Computing Machinery* 9:61–63, 1962.
3. D. Binkele-Raible. *Amortized Analysis of Exponential Time- and Parameterized Algorithms: Measure & Conquer and Reference Search Trees.* PhD thesis, Universität Trier, Germany, 2010.
4. D. Binkele-Raible, L. Brankovic, M. Cygan, H. Fernau, J. Kneis, D. Kratsch, A. Langer, M. Liedloff, M. Pilipczuk, P. Rossmanith, and J. O. Wojtaszczyk. Breaking the 2^n-barrier for Irredundance: two lines of attack. *Journal of Discrete Algorithms*, 9:214–230, 2011.

[9] 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 [3]. Some of these results appeared in several papers, e.g., in [4, 5, 42].

5. D. Binkele-Raible and H. Fernau. Enumerate and measure, improving parameter budget management. In *Parameterized and Exact Computation, IPEC*, volume 6478 of *Lecture Notes in Computer Science*, pages 38–49. Springer, 2010. Long version accepted to be published in *Algorithmica* under the title "Parameterized Measure & Conquer for problems with no small kernels."
6. A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto: Fourier meets Möbius: fast subset convolution. In D. S. Johnson and U. Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, STOC*, pages 67–74. ACM Press, 2007.
7. A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. The Travelling Salesman Problem in bounded degree graphs. In In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 198–209. Springer, 2008.
8. A. Björklund. Determinant sums for undirected Hamiltonicity. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 173–182. IEEE Computer Society, 2010.
9. L. Sunil Chandran and F. Grandoni. Refined memorization for vertex cover. *Information Processing Letters*, 93:125–131, 2005.
10. J. Chen, I. A. Kanj, and W. Jia. Vertex cover: further observations and further improvements. *Journal of Algorithms*, 41:280–301, 2001.
11. J. Chen, I. A. Kanj, and G. Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40–42):3736–3756, 2010.
12. 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. *Journal of Computer and System Sciences*, 76(7):650–662, 2010.
13. D. Eppstein. The Traveling Salesman problem for cubic graphs. In *Journal of Graph Algorithms and Applications* 11(1), pages 61–81, 2007.
14. M. R. Fellows. Towards fully multivariate algorithmics: some new results and directions in parameter ecology. In J. Fiala, J. Kratochvíl, and M. Miller, editors, *Combinatorial Algorithms, 20th International Workshop, IWOCA*, volume 5874 of *Lecture Notes in Computer Science*, pages 2–10. Springer, 2009.
15. H. Fernau, S. Gaspers, and D. Raible. Exact and parameterized algorithms for MAX INTERNAL SPANNING TREE. In C. Paul and M. Habib, editors, *Graph-Theoretic Concepts in Computer Science, 35th International Workshop, WG*, volume 5911 of *Lecture Notes in Computer Science*, pages 100–111. Springer, 2010.
16. 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).
17. H. Fernau, J. Kneis, D. Kratsch, A. Langer, M. Liedloff, D. Raible, and P. Rossmanith. An exact algorithm for the Maximum Leaf Spanning Tree problem. In J. Chen and F. V. Fomin, editors, *Parameterized and Exact Computation, 4th International Workshop, IWPEC, Proceedings*, volume 5917 of *Lecture Notes in Computer Science*, pages 161–172. Springer, 2009. Long version to appear in *Theoretical Computer Science*.
18. F. V. Fomin, S. Gaspers, A. V. Pyatkin, and I. Razgon. On the Minimum Feedback Vertex Set problem: exact and enumeration algorithms. *Algorithmica*, 52(2): 293–307, 2008.
19. F. V. Fomin, S. Gaspers, S. Saurabh, and S. Thomassé. A linear vertex kernel for Maximum Internal Spanning Tree. In Y. Dong, D.-Z. Du, and O. H. Ibarra, editors, *Algorithms and Computation, 20th International Symposium, ISAAC*, volume 5878 of *Lecture Notes in Computer Science*, pages 267–277. Springer, 2009.
20. F. V. Fomin, F. Grandoni, and D. Kratsch. A Measure & Conquer approach for the analysis of exact algorithms. *Journal of the Association for Computing Machinery*, 56(5), 2009.
21. F. V. Fomin, F. Grandoni, and D. Kratsch. Solving Connected Dominating Set faster than 2^n. *Algorithmica*, 52(2):153–166, 2008.
22. F. V. Fomin and D. Kratsch. *Exact Exponential Algorithms*, Springer, 2010.
23. F. V. Fomin, D. Lokshtanov, F. Grandoni, and S. Saurabh. Sharp separation and applications to exact and parameterized algorithms. In A. López-Ortiz, editor, *Proceedings of LATIN, 9th Latin American Theoretical Informatics Symposium*, volume 6034 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 2010.
24. S. Gaspers. *Exponential Time Algorithms: Structures, Measures, and Bounds*. PhD thesis, University of Bergen, 2008.

25. S. Gaspers, S. Saurabh, and A. A. Stepanov. A moderately exponential time algorithm for Full Degree Spanning Tree. In M. Agrawal, D.-Z. Du, Z. Duan, and A. Li, editors, *Proceedings of TAMC, Theory and Applications of Models of Computation, 5th International Conference*, volume 4978 of *Lecture Notes in Computer Science*, pages 479–489. Springer, 2008.
26. S. Gaspers and G. B. Sorkin. A universally fastest algorithm for Max 2-Sat, Max 2-CSP, and everything in between. In C. Mathieu, editor, *Proceedings of SODA, 20th annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, pp. 606–615. Long version accepted to be published in *Journal of Computer and System Sciences*.
27. M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics* 10, pages 196–210, 1962.
28. K. Iwama and T. Nakashima. An improved exact algorithm for cubic graph TSP. In G. Lin, editor, *Computing and Combinatorics, 13th Annual International Conference, COCOON, Proceedings*, volume 4598 of *Lecture Notes in Computer Science*, pages 108–117. Springer, 2007.
29. R. M. Karp. Dynamic programming meets the principle of inclusion-exclusion. *Information Processing Letters*, 1(2):49–51, 1982.
30. M. Knauer and J. Spoerhase. Better approximation algorithms for the Maximum Internal Spanning Tree Problem. In F. K. H. A. Dehne, M. L. Gavrilova, J.-R. Sack, and C. D. Tóth, editors, *Proceedings of WADS, Workshop on Algorithms and Data Structures*, volume 5664 of *Lecture Notes in Computer Science*, pages 459–470. Springer, 2009.
31. S. Kohn, A. Gottlieb, and M. Kohn. A generating function approach to the Traveling Salesman Problem. In *Proceedings of the 1977 ACM Annual Conference*, pages 294–300. ACM Press, 1977.
32. D. Lokshtanov and J. Nederlof. Saving space by algebraization. In L. J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC*, pages 321–330. ACM Press, 2010.
33. D. Lokshtanov and S. Saurabh. Even faster algorithm for Set Splitting! In J. Chen and F. V. Fomin, editors, *Parameterized and Exact Computation, 4th International Workshop IWPEC, Proceedings*, volume 5917 of *Lecture Notes in Computer Science*, pages 288–299. Springer, 2009.
34. L. M. Mays. *Water Resources Engineering*, 2nd ed. Wiley, 2010.
35. J. W. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3:23–28, 1965.
36. J. Nederlof. Fast polynomial-space algorithms using Möbius inversion: improving on Steiner Tree and related problems. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. E. Nikoletseas, and W. Thomas, editors, *Automata, Languages and Programming, 36th International Colloquium, ICALP, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5555 of *Lecture Notes in Computer Science*, pages 713–725. Springer, 2009.
37. J. Nederlof and J. M. M. van Rooij. Inclusion/exclusion branching for Partial Dominating Set and Set Splitting. In V. Raman and S. Saurabh, editors, *Parameterized and Exact Computation — 5th International Symposium, IPEC*, volume 6478 of *Lecture Notes in Computer Science*, pages 204–215. Springer, 2010.
38. K. Ozeki and T. Yamashita. Spanning Trees: A Survey. *Graphs and Combinatorics*, 27:1–26, 2011.
39. E. Prieto. *Systematic Kernelization in FPT Algorithm Design.* PhD thesis, The University of Newcastle, Australia, 2005.
40. E. Prieto and C. Sloper. Either/or: Using vertex cover structure in designing FPT-algorithms—the case of k-internal spanning tree. In F. K. H. A. Dehne, J.-R. Sack, and M. H. M. Smid, editors, *Proceedings of WADS, Workshop on Algorithms and Data Structures*, volume 2748 of *Lecture Notes in Computer Science*, pages 465–483. Springer, 2003.
41. E. Prieto and C. Sloper. Reducing to independent set structure – the case of k-internal spanning tree. *Nordic Journal of Computing*, 12(3): 308–318, 2005.
42. D. Raible and H. Fernau. An amortized search tree analysis for k-Leaf Spanning Tree. In J. van Leeuwen, A. Muscholl, D. Peleg, J. Pokorný, and B. Rumpe, editors, *SOFSEM: Theory and Practice of Computer Science*, volume 5901 of *Lecture Notes in Computer Science*, pages 672–684. Springer, 2010.
43. J. M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7(3):425–440, 1986.
44. G. Salamon and G. Wiener. On finding spanning trees with few leaves. *Information Processing Letters*, 105(5): 164–169, 2008.
45. G. Salamon. Approximation algorithms for the maximum internal spanning tree problem. *Theoretical Computer Science*, 410(50): 5273–5284, 2009.

46. G. Salamon. A survey on algorithms for the maximum internal spanning tree and related problems. *Electronic Notes in Discrete Mathematics*, 36:1209–1216, 2010.
47. M. Wahlström. *Algorithms, Measures and Upper Bounds for Satisfiability and Related Problems*. PhD thesis, Linköpings universitet, Sweden, 2007.