Algorithmica
DOI 10.1007/s00453-007-9115-5

What Would Edmonds Do? Augmenting Paths and Witnesses for Degree-Bounded MSTs

Kamalika Chaudhuri · Satish Rao · Samantha Riesenfeld · Kunal Talwar

Received: 7 September 2007 / Accepted: 1 October 2007

Abstract Given a graph and degree upper bounds on vertices, the BDMST problem requires us to find a minimum cost spanning tree respecting the given degree bounds. This problem generalizes the Travelling Salesman Path Problem (TSPP), even in unweighted graphs, and so we expect that it is necessary to relax the degree constraints to get efficient algorithms. Könemann and Ravi (Proceedings of the Thirty Second Annual ACM Symposium on Theory of Computing, pp. 537–546, 2000; Proceedings of the Thirty-Fifth ACM Symposium on Theory of Computing, pp. 389–395, 2003) give bicriteria approximation algorithms for the problem using local search techniques of Fischer (Technical Report 14853, Cornell University, 1993). Their algorithms find solutions which make a tradeoff of the approximation factor for the cost of the resulting tree against the factor by which degree constraints are violated. In particular, they give an algorithm which, for a graph with a spanning tree of cost C and degree B, and for parameters $b, w > 1$, produces a tree whose cost is at most wC and whose degree is at most $\frac{w}{w-1}bB + \log_b n$.

A primary contribution of Könemann and Ravi is to use a Lagrangean relaxation to formally relate the BDMST problem to what we call the MDMST problem, which is the problem of finding an MST of minimum degree in a graph. In their solution to the MDMST problem, they make central use of a local-search approximation algorithm of Fischer.

K. Chaudhuri (✉) · S. Rao · S. Riesenfeld
University of California, Berkeley, USA
e-mail: kamalika@cs.berkeley.edu

S. Rao
e-mail: satishr@cs.berkeley.edu

S. Riesenfeld
e-mail: samr@cs.berkeley.edu

K. Talwar
Microsoft Research, Mountain View, CA, USA
e-mail: kunal@microsoft.com

In this paper, we give the first approximation algorithms for the BDMST problem—both our algorithms find trees of *optimal* cost. We achieve this improvement using a primal-dual cost bounding methodology from Edmonds' weighted matching algorithms which was not previously used in this context. In order to follow Edmonds' approach, we develop algorithms for a variant of the MDMST problem in which there are degree *lower bound* requirements. This variant may be of independent interest; in particular, our results extend to a generalized version of the BDMST problem in which both upper and lower degree bounds are given.

First we give a polynomial-time algorithm that finds a tree of *optimal* cost and with maximum degree at most $\frac{b}{2-b}B + O(\log_b n)$ for any $b \in (1, 2)$. We also give a quasi-polynomial-time approximation algorithm which produces a tree of optimal cost C and maximum degree at most $B + O(\log n / \log\log n)$. That is, the error is additive as well as restricted just to the degree. This further improvement in degree is obtained by using augmenting-path techniques that search over a larger solution space than Fischer's local-search algorithm.

Keywords Approximation algorithms · Combinatorial optimization · Minimum spanning trees · Matching

1 Introduction

We study the problem of finding a minimum cost tree among all spanning trees of a graph $G = (V, E)$ that meet degree bounds on the vertices in V. A solution to this problem, called a degree-bounded minimum spanning tree (BDMST), can be used to find a minimum cost travelling salesman path (TSPP) between two vertices u and v of G by finding the BDMST where u and v have degree bounds of one and other nodes have degree bounds of two. Since the NP-hard problem of computing a Hamiltonian path can be solved given even a polynomial-factor approximation algorithm for TSPP, approximating a TSPP to within a polynomial factor is NP-hard. Therefore, assuming that P does not equal NP, it is necessary to relax the degree bounds to get an efficient approximation algorithm for the BDMST problem.

If the edge weights in the graph obey the triangle inequality, Hoogeveen [12] provides an algorithm for the TSPP problem with an approximation ratio of $3/2$; note that we do not make any such assumptions and work with general edge weights.

The BDMST problem has several applications [21] in efficient network routing protocols. For example, in multicast networking, the objective may be to build a multicast network that minimizes both the total cost of the network and the maximum work done by a router. In graph theoretic terms, this translates to finding a spanning tree that minimizes both cost and degree. The BDMST problem is a natural formulation of these competing constraints.

For the sake of simplicity, we focus on what we call the uniform version of the problem, in which each node has the same degree bound B, though our results generalize to nonuniform degree bounds.

Bicriteria approximation algorithms for this problem were first given by Ravi et al. [21, 22]. The best previous results are by Könemann and Ravi [16, 17]. They give an algorithm that, for a graph with a BDMST of maximum degree B and cost C, and for any constants $b, w > 1$, finds a spanning tree with maximum degree $\frac{w}{w-1}bB + \log_b n$ and cost wC. While this expression yields a variety of tradeoffs, one can observe that the Könemann -Ravi algorithm always either violates the degree constraints or exceeds the optimal cost by a factor of two.

We give a true polynomial-time approximation algorithm for the BDMST problem that removes the error in the cost completely, as well as a quasi-polynomial-time approximation algorithm that both obtains a solution of optimal cost and significantly improves the error in the degree bounds. Specifically, given a graph with a BDMST of optimal cost C and maximum degree B, our polynomial-time algorithm finds a spanning tree of cost exactly C and maximum degree $\frac{b}{2-b}B + O(\log_b n)$ for any constant $b \in (1, 2)$. The main technical contribution here is the use of a subroutine that finds a MST of minimum degree which also respects certain lower bounds on degrees. This allows for more natural and better performing cost-bounding techniques, which we discuss below.

Our quasi-polynomial-time approximation algorithm for the BDMST problem finds a spanning tree of maximum degree $B + O(\frac{\log n}{\log\log n})$ and optimal cost C. If the degree is $\omega(\log n/\log\log n)$, our algorithm gives a $1 + o(1)$ approximation in the degree while achieving optimal cost. In addition to the cost bounding techniques already described, the main technical contribution here is the use of augmenting paths to find low-degree MSTs as opposed to the local-search techniques used by Könemann and Ravi [16] to solve this subproblem.

In a recent paper, Könemann and Ravi [17] improve their algorithms both in terms of implementation and applicability by providing an algorithm that can deal with nonuniform degree bounds and does not rely on explicitly solving a linear program. Our polynomial-time algorithm works as does theirs for nonuniform bounds, except it achieves cost optimality. Our quasi-polynomial-time algorithm extends even more easily to nonuniform degree bounds since the error is additive, rather than multiplicative.

The algorithms that we give for enforcing lower bounds on degree require new ideas and may also be of independent interest. Our results extend to a more general version of the BDMST problem, in which upper bounds *and lower bounds* on the degree of vertices are given.

1.1 Previous Techniques

Fürer and Raghavachari [9] give a beautiful polynomial-time algorithm for unweighted graphs reminiscent of Edmonds' algorithm for unweighted matching [5] that finds a spanning tree of degree $\Delta + 1$ in any graph that has a spanning tree of optimal degree Δ. This is optimal, in the sense that an exact algorithm would give an algorithm for Hamiltonian path and is therefore not expected to run in polynomial-time. Their algorithm takes any spanning tree and relieves the degree of a high-degree node by an alternating sequence of edge additions and deletions. While there is a node in the current tree of degree at least $\Delta + 1$, the algorithm either finds such a way to lower its degree or certifies that the maximum degree of *any* spanning tree must be at least Δ. The certificate consists of a set of nodes S whose removal leaves at least

$\Delta|S|$ connected components in the graph. This implies that the average degree of the set S must be at least Δ.

For the weighted version of the BDMST problem, we consult a different algorithm of Edmonds [4] that computes *weighted* matchings. This algorithm can be viewed as first finding a solution to the dual of the matching problem: it finds an assignment of penalties to nodes that can be thought of as increasing the cost of adjacent edges, and then it finds a maximal packing of dual variables for subsets of nodes of the graph. Once the dual solution is known, one can ignore the values of the weights on the tight edges and rely solely on an un-weighted matching algorithm.

Könemann and Ravi take an approach similar to Edmonds by examining a linear programming relaxation for the BDMST problem and its dual.[1] However in this case, the combinatorial subroutine must produce an MST that has minimum maximum degree over all MSTs in the graph where the edge costs have been modified by the dual penalties. We call this subproblem the MDMST problem.

To find an MDMST in the modified graph, Ravi and Könemann use an algorithm of Fischer [8] that for a graph with an MST of degree Δ, provides an MST of degree at most $b\Delta + \log_b n$ for any $b > 1$. Könemann and Ravi show how to get a good BDMST solution by solving a linear program with a relaxed (by a factor of $(w/w-1)$) degree constraint, using the solution to get the modified edge costs and then applying Fischer's algorithm to obtain an approximate MDMST of the modified graph.

1.2 Our Results and Techniques

As mentioned above, Könemann and Ravi's methods introduce a tradeoff between degree and cost that always creates a constant-factor error either in the degree or in the cost.

In this paper, we remedy this problem by following further Edmonds' approach to weighted matchings. We show that any MST of the modified graph in which the nodes with dual penalties have (relatively) high degree has low cost in the original graph. Thus, instead of just finding an MST of low maximum degree, we also wish to force the nodes with penalties to have high degree. Accomplishing this entails developing algorithms and certificates for enforcing lower bounds on degrees, in addition to our algorithms for enforcing upper bounds. Moreover, we have to carefully combine the algorithms to simultaneously enforce the high- and low-degree constraints.

In Sect. 4, we present a polynomial-time algorithm for MDMST based on Fischer's algorithm which enforces both upper and lower bounds on degrees. Specifically, given a graph G and a lower bound B' on a specified set of vertices S, our algorithm finds a MST of G which has degree at most $b\Delta + \log_b n$ and in which the nodes in S have degree at least $B'/b - \log_b n$. Here Δ is the minimum degree of any MST that respects the lower degree bounds on S. Combining this algorithm with our cost bounding techniques gives us the polynomial-time algorithm for BDMST.

[1]As in the case of Edmonds' non-bipartite matching algorithm, the linear program and its dual are of exponential size, though their solutions can be found in polynomial time.

In addition to improving the cost of the solution, we improve the degree-bounds at the cost of a quasi-polynomial running time. Our approach relies on our analogy with bipartite matching. As a guide to our approach, consider the problem on bipartite graphs of assigning each node on the "left" to a node on the "right" while minimizing the maximum degree of any node on the right. This problem can naturally be solved using matching algorithms. Consider instead finding a locally optimal solution where for each node on the left, all of its neighbors have degree within 1 of each other. One can then prove that no node on the right has degree higher than $b\Delta + \log_b n$ where Δ is the optimum value. The proof uses Hall's lower bound to show that any breadth first search in the graph of matched edges is shallow, and the depth of such a tree bounds the maximum degree of this graph. This is the framework of Fischer's algorithm for finding a minimum cost spanning tree of degree $b\Delta + \log_b n$.

For the bipartite graph problem above, an augmenting- or alternating-path algorithm for matching gives a much better solution than the locally greedy algorithm. Based on this insight, we use augmenting paths to find a MDMST of degree at most $\Delta + O(\frac{\log n}{\log\log n})$, where Δ is the optimal degree over all MSTs. In contrast to Fisher's local (or single-move) approach, our algorithm finds a sequence of moves to decrease the degree of one high-degree node. This introduces significant complications since changing the structure of the tree also changes the permissible moves.

We also remark that our augmenting path algorithms, though based on the most powerful methods in matching, fall short of the $\Delta + 1$ that Fürer and Raghavachari obtain.

1.3 Recent Work

Subsequent to the publication of this work [2], there has been significant progress on the problem. The authors [3] gave the first constant factor approximation to the MDMST problem on weighted graphs. The algorithm in [3] differs significantly in techniques from the present work—it uses the push-relabel framework and outputs an MST of degree at most $2\Delta_{\mathrm{OPT}} + O(\sqrt{\Delta_{\mathrm{OPT}}})$. For the BDMST problem, this gives a spanning tree with degree at most $\frac{2w}{w-1}B + O(\sqrt{wB/(w-1)})$ and cost wC in a graph that has a spanning tree of cost C and degree B. Ravi and Singh [20] presented an algorithm that outputs an MST of degree at most $\Delta_{\mathrm{OPT}} + k$, where k is the number of distinct edge weights in the graph. Goemans [11] recently gave an algorithm that outputs an MST of degree at most $\Delta_{\mathrm{OPT}} + 2$, nearly matching the result of Fürer and Raghavachari for the unweighted case. Progress finally culminated in an algorithm by Singh and Lau [23] that outputs an MST of degree $\Delta_{\mathrm{OPT}} + 1$. The algorithms of Goemans and of Singh and Lau use polyhedral techniques and work also for the BDMST problem, outputting trees of degree at most $B + 2$ and $B + 1$, respectively, and of cost at most that of the minimum-cost tree of degree B.

2 Bounded-Degree Minimum Spanning Trees

BDMST Problem Given a weighted graph $G = (V, E, c)$, $c : E \to R^+$, and a positive integer $B \geq 2$, find the minimum cost spanning tree in the set $\{T : \forall v \in V, \deg_T(v) \leq B\}$.

Könemann and Ravi [17] show that an MST of minimum degree for a certain cost function is also a BDMST with analogous guarantees on degree and with cost within a constant factor of the optimal. We show in this section that an algorithm which obtains an MST with both guaranteed upper and lower degree bounds can be used to produce a BDMST with optimal cost and analogous degree bounds. We present two algorithm in Sects. 4 and 5 that obtain such guarantees.

2.1 Linear Programs for BDMST

An integer linear program for the BDMST problem is given by:

$$\begin{aligned} \text{OPT}_B \quad = \quad \min \quad & \sum_{e\in E} c_e x_e \\ \text{s.t.} \quad & \sum_{e\in\delta(v)} x_e \le B \quad \forall v \in V, \\ & x \in \text{SP}_G, \\ & x_e \in \{0,1\}, \end{aligned} \tag{1}$$

where $\delta(v)$ is the set of edges of E that are incident to v, and SP_G is the convex hull of edge-incidence vectors of spanning trees of G. A vector $x \in \text{SP}_G$, the entries of which are not necessarily all integer, is called a *fractional* spanning tree. It can be written as a convex combination of spanning trees of G. In general we use the notation τ and T to refer to a fractional spanning tree and a strictly integral spanning tree, respectively. For a fractional tree τ with edge incidence vector $x \in \text{SP}_G$, let $\deg_\tau(v) = \sum_{e\in\delta(v)} x_e$. Now we can write the linear program relaxation of (1) by replacing the integrality constraint by $0 \le x_e \le 1$. Because of the integrality of the spanning tree polytope, we can rewrite the relaxation as:

$$\begin{aligned} \min \quad & \sum_{T\in E} c(T)\alpha_T \\ \text{s.t.} \quad & \sum_T \deg_T(v)\alpha_T \le B \sum_T \alpha_T \quad \forall v \in V, \\ & \sum_T \alpha_T = 1, \\ & \alpha_T \ge 0, \end{aligned} \tag{2}$$

where we have a variable α_T for each spanning tree T of G, i.e. for each vertex of SP_G. Now one can write the dual of this linear program:

$$\begin{aligned} \text{OPT}_{LD(B)} \quad = \quad \max \quad & \nu \\ \text{s.t.} \quad & \nu \le c(T) + \textstyle\sum_{v\in V} \lambda_v(\deg_T(v) - B) \quad \forall T, \\ & \lambda_v \ge 0 \quad \forall v \in V. \end{aligned} \tag{3}$$

This linear program, though it has exponentially many constraints, can be solved in polynomial time; in fact it is equivalent to the Lagrangean dual of (1) as used by

Könemann and Ravi [17]:

$$\max_{\lambda \geq 0} \min_{\tau \in \mathrm{SP}_G} \left\{ c(\tau) + \sum_{v \in V} \lambda_v (\deg_\tau(v) - B) \right\}. \tag{4}$$

Let (λ^B, α_T) be a pair of optimal solutions to the primal and dual. Following the analysis of [17], let us define a new cost function c^{λ^B}, where $c^{\lambda^B}(e) = c_e + \lambda_u^B + \lambda_v^B$ for an edge $e = (u, v)$. Let $\mathcal{O}^B$ be the set $\{T : \alpha_T > 0\}$ of spanning trees with positive α_T values. The complementary slackness conditions then imply that every T in $\mathcal{O}^B$ minimizes $c(T) + \sum_{v \in V} \lambda_v (\deg_T(v) - B)$. Since $\sum_{v \in V} \lambda_v B$ is independent of T, we conclude that each tree in $\mathcal{O}^B$ minimizes $c(T) + \sum_{v \in V} \lambda_v \deg_T(v) = c^{\lambda^B}(T)$; i.e. every tree in $\mathcal{O}^B$ is a minimum spanning tree under the cost function c^{λ^B}. The optimal solution to the linear program is then a fractional MST (again under c^{λ^B}): $\tau^B = \sum_{T \in \mathcal{O}^B} \alpha_T T$. Note that the degree of v in τ^B is $\deg_{\tau^B}(v) = \sum_T \alpha_T \deg_T(v)$.

Let $L^B = \{v : \lambda_v^B > 0\}$ be the set of nodes with positive dual variables in the optimal solution. Complementary slackness conditions then imply the following claim.

Lemma 1 *For all* $v \in V$, $\deg_{\tau^B}(v) \leq B$, *and for all* $v \in L^B$, $\deg_{\tau^B}(v) = B$.

So we are given the existence of a fractional BDMST τ of G such that τ is in fact a fractional MST under the cost function c^{λ^B} and it meets both upper and lower degree bounds (i.e. no node in τ has degree more than B and every node in L^B has degree exactly B). Our approach to approximating a solution to the BDMST problem is to find an *integral* MST, under a slightly different cost function, that approximately meets both the upper and lower degree bounds. Then we use the dual program to show that this tree does not cost too much more than the cost of τ.

More precisely, let $B^* = B + \omega$, for some $\omega > 0$ to be specified later. Let $T^{B^*} \in \mathcal{O}^{B^*}$, so T^{B^*} is an MST under the cost function $c^{\lambda^{B^*}}$. Let L^{B^*} be the set of nodes with positive dual variables in the optimal fractional LP solution with degree bound B^*. Since λ^{B^*} is a feasible solution for the dual LP (4), it is clear that

$$c(T^{B^*}) + \sum_{v \in V} \lambda_v^{B^*} (\deg_{T^{B^*}}(v) - B) \leq \mathrm{OPT}_{LD(B)}.$$

Further, if T^{B^*} has the property that for every node $v \in L^{B^*}$, $\deg_{T^{B^*}}(v)$ is at least $B = B^* - \omega$, the second term in the above expression is non-negative and hence $c(T^{B^*})$ is at most $\mathrm{OPT}_{LD(B)}$. To summarize:

Lemma 2 *Let T be an MST of G under the cost function $c^{\lambda^{B^*}}$ such that for every $v \in L^{B^*}$, $\deg_T(v) \geq B$. Then $c(T) \leq \mathrm{OPT}_{LD(B)}$.*

3 Algorithmic Preliminaries

3.1 Swaps

Let e be an edge in an MST T. For some edge $e' \in E$, we say that the pair (e, e') is a *swap* with respect to T, or that e and e' may be *swapped*, if the following conditions hold: (a) $e' \notin T$, (b) the unique cycle in $T \cup e'$ contains e, and (c) e and e' have the same cost. When we perform the swap (e, e'), we remove e from T and add e' to produce another MST T'.

For a tree T, let $d_{\max}(T)$ denote the maximum degree in T over all vertices, and let $d^L_{\min}(T)$ denote the minimum degree over all vertices of L in T. For an integer $d > 0$ and a tree T, let $S_{\geq d}$ be the set of nodes with degree at least d in T, and let $S^L_{\leq d}$ be the subset of nodes in L that have degree at most d.

We shall also repeatedly use the *exchange property* of the minimum spanning tree matroid. This matroid property states that for any two MSTs T, T', and for any edge $e' \in T'$ such that $e' \notin T$, there exists an edge $e \in T$ such that $e \notin T'$ and (e, e') is a swap. Moreover, for any edge $e \in T$ such that $e \notin T'$, there is an edge $e' \in T'$ such that (e, e') is a swap with respect to T. Additionally, the exchange property continues to hold if we force T, T' to contain a set F of edges, and exclude a set R of edges; in this case e (respectively e') lies outside $F \cup R$ if e' (respectively e) does.

3.2 Witnesses

As observed in Sect. 2.1, the problem of finding a BDMST can be reduced to the problem of finding an actual MST, under a different cost function, that satisfies upper and lower bounds on the degrees of nodes. In Sect. 4, we give a polynomial-time algorithm that find an MST respecting given upper bounds and another polynomial-time algorithm that finds an MST respecting given lower bounds. The algorithms are then carefully combined to achieve a polynomial-time algorithm for the BDMST problem. In later sections, we show how to replace these sub-routines with algorithms that achieve much better approximations in degree but may take quasi-polynomial time.

An important aspect of all of these algorithms is that when they terminate, they produce a proof that the resulting tree has close-to-optimal degree. We call such a combinatorial proof a *witness*. We first introduce notation for a decomposition of the graph G that forms the basis for the witnesses produced by all our algorithms. The decomposition is a partition of the nodes of G into a *center set* W and k other sets $C_1, C_2, \ldots, C_k$ called *clusters*. For a tree T and a cluster $C \subseteq V$, we say C is *internally connected* in T if the induced subgraph (C, T_C) is connected, where $T_C = \{(u, v) \in T : u, v \in C\}$.

In the case of the high-degree witness for the MaxdmstP and MaxdmstQ algorithms, the partition has the property that each cluster has at least one tree edge to W. In fact, it has the stronger property that in *any* MST of G, there is at least one edge from C_i to W. See Fig. 1.

If it is also true that k is large, then the average degree of W in any MST must be high. The following lemma formalizes this idea. A similar lemma was used by [9].

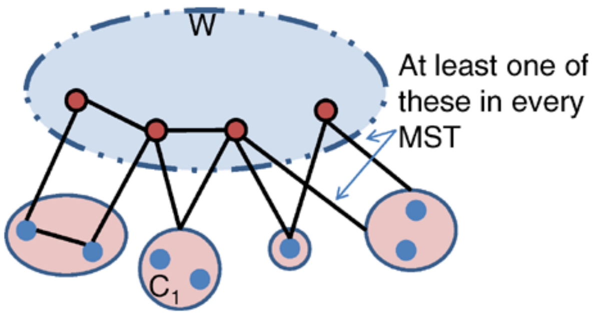

Fig. 1 Example of a simple high-degree witness, as used in Lemma 3

Lemma 3 *Let V be partitioned into sets of nodes $W, C_1, C_2, \ldots, C_k$. If for every cluster C_i any MST contains an edge connecting C_i to W, then for any MST T of G, $d_{\max}(T) \geq \lceil \frac{k}{|W|} \rceil$.*

Proof Since the C_i's are disjoint and each one contains an edge from C_i to W, the total degree of W is at least k. Since the maximum degree in W is larger than the average, the claim follows. □

We note that the actual witnesses output by the MaxdmstP and MindmstP algorithms are slightly more complicated; see Sects. 4.1 and 5.1 for details. The algorithms respecting degree lower bounds (MindmstP and MindmstQ) also output witnesses of a similar flavor, though the structure is somewhat different; see Sects. 4.2 and 5.2 for details.

4 MSTs with Degree Bounds : A Polynomial Time Algorithm

We formally define the MSTDB problem as follows.

MSTDB Problem Given a graph $G = (V, E)$ with costs on edges, a degree upper bound B_H, a subset of vertices $L \subseteq V$, and a degree lower bound B_L, find, if one exists, a minimum spanning tree of G such that no vertex has degree more than B_H and all vertices in L have degree at least B_L.

Note that an MST with these degree bounds may not exist in the graph. In this case, we would like an algorithmic solution to provide a proof of non-existence.

Our polynomial-time algorithm for this problem is based on Fischer's algorithm for finding a minimum-degree MST of a graph. Our algorithm finds an MST such that (a) every vertex has degree at most $bB_H + \log_b n$ and (b) all vertices in L have degree at least $B_L/b - \log_b n$. If it fails, it finds a combinatorial witness to show that there exists no MST of the graph in which all vertices have degree at most B_H and the vertices in L have degree at least B_L.

Starting with an arbitrary MST, our MstdbP algorithm proceeds in phases of improvement. Each phase is essentially a combination of the two algorithms MaxdmstP and MindmstP, which are described in Sects. 4.1 and 4.2. The MaxdmstP algorithm is essentially Fischer's algorithm, and MindmstP is a symmetric version of Fischer's algorithm that finds an MST in which a given set of nodes have maximum minimum degree.

To measure the progress made by our algorithm, we define a potential function $\phi(T)$ on the set of all MSTs. The potential function has two components: $\phi_H(T)$ and $\phi_L(T)$. The potential function $\phi_H(T)$ decreases as we move towards a tree in which more vertices satisfy a degree upper bound of B_H, and the potential function $\phi_L(T)$ decreases as we move towards a tree in which more vertices in L satisfy the degree lower bound of B_L. The potential functions are defined as follows:

$$\phi_H(T) = \sum_v 5^{\deg(v)-B_H},$$

$$\phi_L(T) = \sum_{v \in L} 5^{B_L-\deg(v)},$$

$$\phi(T) = \phi_H(T) + \phi_L(T).$$

Note that for $d > B_H$, performing a swap that decreases the number of vertices with degree at least d in a tree T decreases the potential $\phi_H(T)$, since the swap does not increase the number of vertices with degree at most $d-1$ by more than 2. Similarly, for $d < B_L$, the potential $\phi_L(T)$ decreases if a swap is performed that decreases the number of vertices in L with degree at most d, since the swap does not increase the number of vertices in L with degree at least $d+1$ by more than 2.

4.1 The MaxdmstP Algorithm

Given an MST T of G, let $\mathcal{T} \subseteq T$ be a Steiner tree on the nodes of $S_{\geq(d-1)}$ (the Steiner nodes are the nodes in $V \setminus S_{\geq(d-1)}$). We say that we *freeze* the edges of $\mathcal{T}$ incident on $S_{\geq(d-1)}$, meaning that the edges of $\mathcal{T}$ that are incident on $S_{\geq(d-1)}$ are not allowed to be removed by any swap. For a subset $X \subseteq V$, an integer $d > 0$, and a tree T on G, we call a swap (e, e') (X, d)*-deflating* if e is incident on $S_{\geq d}$ but not on X, and e' is not incident on $S_{\geq(d-1)}$. In other words, the swap (e, e') decreases the degree of a node in $S_{\geq d}$ without increasing the degree of a node in $S_{\geq d-1}$ or decreasing the degree of a node in X. We look for (X, d)-deflating swaps (e, e') where e is not frozen.

Given an MST T, a set of nodes X and an integer d, the MaxdmstP algorithm is very simple: first freeze the edges of $\mathcal{T}$ incident on $S_{\geq(d-1)}$ as above. Find and execute a (X, d)-deflating swap (e, e') where e is not frozen, if it exists. See Fig. 2 for a formal description.

When the MaxdmstP and MindmstP algorithms are combined later, X corresponds to the set of low-degree nodes. Since we do not want the low-degree nodes to lose degree, we disallow deletion of edges incident on them. The steiner tree $\mathcal{T}$ is frozen to help create the witness when the algorithm fails to find a swap.

To show that the MaxdmstP algorithm works as promised, we need to use a somewhat more complicated witness than the one described in Sect. 3.2. A witness $\mathcal{W}$ consists of a partition of the nodes into a center set U and clusters $C_1, C_2, \ldots, C_k$, a set of nodes $W \supseteq U$, and a set of frozen edges F. The witness $\mathcal{W}$ must have the

```
Algorithm MaxdmstP($G, T, X, d$)

    Let $\mathcal{T} \subseteq T$ be a Steiner tree on $S_{\geq(d-1)}$. Freeze the edges of $\mathcal{T}$ incident
    on $S_{\geq(d-1)}$.
    if there is an $(X, d)$-deflating swap $(e, e')$ where $e$ is not frozen
        then output $T \setminus \{e\} \cup \{e'\}$.
```

Fig. 2 Pseudo-code for MaxdmstP

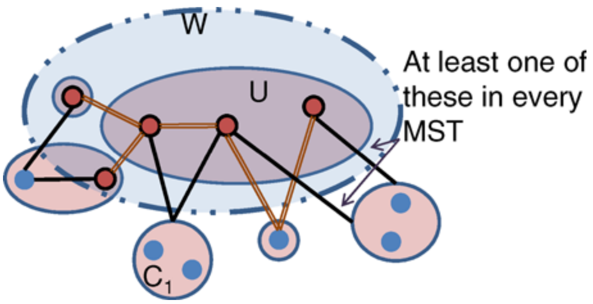

Fig. 3 Example of a high-degree witness, as used in Lemma 4. The double edges are frozen

property that any MST that includes the edges in F must contain at least one edge from each C_i to W. Thus this witness is similar to the witness described earlier in Sect. 3.2 except for the set F of frozen edges. See Fig. 3.

Intuitively, W is a set of high-degree vertices, and U is a set of "incurably" high-degree nodes; i.e. their degree cannot be decreased without increasing the degree of W. The set F of frozen edges is useful technically in the proofs. Now if the size of F is small and k is large, then any MST must use a large number of edges incident to W.

Lemma 4 (Witness to high optimal degree) *A high-degree witness $\mathcal{W} = (\{U, C_1, C_2, \ldots, C_k\}, W, F)$ certifies that any fractional MST τ of G has maximum degree at least*

$$\left(\frac{k - 2|F|}{|W|}\right).$$

Proof First we restrict ourselves to MSTs that contain all the edges in F. The proof in this case is identical to that of Lemma 3 except for the following difference: Unlike the witness in Lemma 3, W may not be disjoint from the C_i's. However any edge that simultaneously connects C_i and C_j to W (i.e. an edge from $C_i \cap W$ to $C_j \cap W$) contributes two units of degree to W. The bound in Lemma 3 therefore continues to hold.

In any MST that does not contain all the edges of F, the total degree of W can be smaller by at most $2|F|$. The average degree of W is therefore at least $(k - 2|F|)/|W|$ in any MST of G. Since a fractional MST is a convex combination of MSTs, the average degree bound also holds for any fractional MST. □

If the MaxdmstP algorithm fails to find a swap, we can create a witness as follows. Let $U = S_{\geq d}$ be the center set. Let $C_1, C_2, \ldots, C_k$ be the connected components obtained by deleting the nodes in U from the current MST T. We set F to be the union of the set of edges of $\mathcal{T}$ incident on $S_{\geq(d-1)}$ and the set of edges in T belonging to $X \times S_{\geq d}$. Finally W is the set $S_{\geq d-1}$. Theorem 5 calculates the guarantees offered by this witness. With some foresight, we assume that X is chosen appropriately.

Theorem 5 ([8]) *Suppose we are given as input an MST T, an integer d and a set of nodes X such that $X = S^L_{\leq(B_L+B_H-d+1)}$. When the algorithm* MaxdmstP *is called on this input, it either outputs a tree with potential function value at most $\phi(T) - 5^{d-B_H-1}$, or finds a witness $\mathcal{W}$ that certifies that for any fractional MST τ of G,*

$$d_{\max}(\tau) \geq (d-3)\frac{|S_{\geq d}|}{|S_{\geq(d-1)}|} - \frac{2|X|}{|S_{\geq(d-1)}|} - 4.$$

Proof The first part of the theorem holds if there is an (X, d)-deflating swap in T. Executing the swap reduces the degree of at least one node in $S_{\geq d}$ and does not increase the degree of any node in $S_{\geq d-1}$. Hence $\phi_H(T)$ decreases by at least $3(5^{d-B_H-1})$. Since a swap decreases the degree of at most two nodes, both outside $X = S^L_{\leq(B_L+B_H-d+1)}$, $\phi_L(T)$ increases by at most $2(5^{B_L-(B_L+B_H-d+1)}) = 2(5^{d-B_H-1})$. The claim follows.

Otherwise let $\mathcal{W} = (\{U, C_1, \ldots, C_k\}, W, F)$ be defined as above. We first argue that $\mathcal{W}$ is a witness. Suppose otherwise, i.e. that there is a tree T' containing F that has no edge going from C_i to W. The proof will show that the existence of such a T' contradicts the failure to find an (X, d)-deflating swap.

Let T_i be the set of edges in T from U to C_i. We first argue that $|T_i \setminus F| \leq 1$. By definition of C_i, it is a connected component of $T \setminus T_i$ and is therefore internally connected in $T \setminus (T_i \setminus F)$. Moreover, $T \setminus (T_i \setminus F)$ contains $\mathcal{T}$ so that U is connected in $T \setminus (T_i \setminus F)$. Thus if there were two edges in $(T_i \setminus F)$ connecting C_i to U, T would have a cycle.

Thus $|T_i \setminus F| \leq 1$. Since T' contains F but has no edges from $U \subseteq W$ to C_i, the set $T_i \cap F$ is empty and T_i contains a single edge, say e_i. The discussion above implies that C_i is a component of $T \setminus \{e_i\}$. By the exchange property, $T \setminus \{e_i\}$ can be completed from T'; that is, there is an edge $e'_i \in T'$ such that (e_i, e'_i) is a swap with respect to T. Since e'_i must be incident on C_i, it is not incident on W and thus (e_i, e'_i) must be an (X, d)-deflating swap. Since no such swaps exist for T, we derive a contradiction.

Finally, note that $|F| \leq 2|S_{\geq(d-1)}| + (|S_{\geq d}| + |X| - 1)$, since the average degree of $S_{\geq(d-1)}$ in the steiner tree $\mathcal{T}$ is at most two, and the second constituent of F is a forest on $S_{\geq d} \cup X$. Moreover, $k \geq (d-2)|S_{\geq d}|$, since each edge from $S_{\geq d}$ to $V \setminus S_{\geq d}$ deleted during the construction of $\mathcal{W}$ contributes an additional component, and the average external degree of $S_{\geq d}$ is at least $(d-2)$. The claim follows from Lemma 4. □

4.2 The MindmstP Algorithm

Here we give an algorithm for the nearly symmetric problem of finding an MST that respects degree lower bounds for a subset L of the vertices.

Fig. 4 Pseudo-code for MindmstP

```
Algorithm MindmstP$(G, T, L, Y, d)$

    if there is an $(Y, d)$-inflating swap $(e, e')$
        then output $T \setminus \{e\} \cup \{e'\}$
     else Create Witness.
```

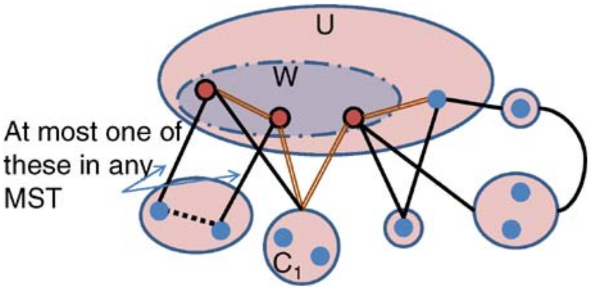

Fig. 5 Example of a low-degree witness, as used in Lemma 6. The double-edges are in H (R and Y are not shown)

For an MST T, subsets of vertices $L, Y \subseteq V$, and an integer $d > 0$, we call a swap (e, e') (Y, d)-*inflating* if e' is incident on $S^L_{\leq d}$ but not on Y and e is not incident on $S^L_{\leq(d+1)}$. That is, an (Y, d)-inflating swap is one that increases the degree of a node in $S^L_{\leq d}$ without increasing the degree of a node in Y or decreasing the degree of a node in $S^L_{\leq(d+1)}$.

The MindmstP algorithm is very natural: given $Y \subseteq V$, $d > 0$, and an MST T, execute an (Y, d)-inflating swap, if one exists. See Fig. 4 for a formal description.

We define the witness $\mathcal{W}_{L,P}$ provided by the MindmstP algorithm as follows. Let V be partitioned into sets $U, C_1, C_2, \ldots, C_k$. Let W be a subset of U, and let $Y \subset V$ be disjoint from U. Let $R \subseteq (Y \times W)$ be a subset of the edges between Y and W. Finally, let H be another set of edges incident on W. A witness $\mathcal{W}_{L,P} = (\{U, C_1, \ldots, C_k\}, W, Y, R, H)$ is a decomposition of the graph such that in any MST T that contains the edges in H and excludes the edges in R, for each i, $1 \leq i \leq k$, there is at most one edge in T between C_i and W that does not belong to H. See Fig. 5 for an illustration.

The witness is in spirit very similar to the high degree witness. Intuitively, U is a set of low degree vertices, and W is a set of "incurably" low-degree nodes in L; i.e. their degree cannot be increased without taking degree away from U. The set Y can be thought of as a set of high-degree nodes and R as a set of edges from high-degree nodes to low-degree nodes; these constituents of the witness are needed later when the MindmstP algorithm is combined with the MaxdmstP algorithm to prevent the MindmstP algorithm from raising the degree of high-degree nodes. The set H of edges is useful technically in the proofs to ensure certain connectivity properties within W.

Lemma 6 *Given a graph $G = (V, E)$, a low-degree witness $\mathcal{W}_{L,P} = (\{U, C_1, \ldots, C_k\}, W, Y, R, H)$ certifies that in every fractional MST τ of G,*

$$d^L_{\min}(\tau) \leq \frac{k + |U| + 2|W| + |Y| + |H| - 2}{|W|}.$$

Proof First let us consider an integral MST T such that T contains H and excludes R. The existence of the witness $\mathcal{W}_{L,P}$ guarantees that for each i, $1 \le i \le k$, T has at most one edge between C_i and W that is not in H. This implies that W cannot have total degree in T more than $k + |U| + |W| - 2$, since of the $(k + |U| - 1)$ edges incident on W, at most $|W| - 1$ can have both endpoints in W. In an arbitrary integral MST, the total degree of W cannot be improved to more than $k + |U| + 2|W| + |Y| + |H| - 2$—since one could replace the $|H|$ edges in H, and add at most $|Y| + |W| - 1$ of the edges from R. Since a fractional MST is just a convex combination of integral MSTs, the upper bound on the total degree of W remains the same, and the claim follows. □

If the MindmstP algorithm fails to make a swap, we create a witness as follows. Let $U = S^L_{\le(d+1)}$, let $W = S^L_{\le d}$, and let $C_1, \ldots, C_k$ be the components that remain when U is removed from T. Choose a Steiner tree $\mathcal{T}$ on U such that $\mathcal{T} \subseteq T$, and let H be the set of edges in $\mathcal{T}$ that are incident on U. Let R be the set of edges of G between Y and W that are *not* in T, that is, $R = (Y \times W) \setminus T$. Let $\mathcal{W}_{L,P} = (\{U, C_1, \ldots, C_k\}, W, Y, R, H)$. The following theorem quantifies the quality of the witness, when the algorithm produces one. With some foresight, we assume the set Y is chosen appropriately.

Theorem 7 *Suppose we are given as input a graph $G = (V, E)$, an MST T, subsets $L, Y \subseteq V$ and an integer $d > 0$. Then the algorithm* MindmstP *when called with $Y = S_{\ge(B_L+B_H-d-1)}$, either produces a tree T with potential function value at most $\phi(T) - 5^{B_L-d-1}$, or finds a witness $\mathcal{W}_{L,P}$ that certifies that for any fractional MST τ of G*

$$d^L_{\min}(\tau) \le (d+3)\frac{|S^L_{\le(d+1)}|}{|S^L_{\le d}|} + \frac{|Y|}{|S^L_{\le d}|} + 2.$$

Proof In the case that the algorithm finds a (Y, d)-inflating swap, we increase the degree of some node in $S^L_{\le d}$, and affect the degree of at most three other nodes by one. Moreover, we do not decrease the degree of any node in $S^L_{\le(d+1)}$. It follows that the potential function decreases by the claimed amount.

Suppose then that the algorithm fails to find a swap. Let T be the tree for which the algorithm creates a witness. We show first that $\mathcal{W}_{L,P} = (\{U, C_1, \ldots, C_k\}, W, Y, R, H)$ defined as above is a witness. Assume otherwise: that is, there is some MST T' containing H and excluding R such that for some i, $1 \le i \le k$, there are two edges e'_1, e'_2 between C_i and W in T' that do not belong to H. (Note that C_i need not be a connected component in T'.) At least one of these two edges, say e'_2, is not in T. By the construction of the clusters, T has at most one edge, say e_i, that is not in H and that is between W and C_i.

The proof argues that the existence of e'_1 and e'_2 in T' contradicts the failure to find a (Y, d)-inflating swap. We consider two cases:

Case 1: $e'_1 \in T$, i.e. $e_i = e'_1$. Since T and T' are both MSTs and $e'_2 \notin T$, there is an edge $e_2 \in T$ different from e'_1 such that (e_2, e'_2) is a swap with respect to T.

Since (e_2, e_2') is not (Y, d)-inflating, e_2 must be incident on $U = S^L_{\le(d+1)}$. Consider $T_2 = T \setminus \{e_2\} \cup \{e_2'\}$. Since e_2 is not internal to C_i, C_i is still internally connected in T_2. Let u_1' and u_2' be the endpoints of e_1' and e_2', respectively, in W. There is a simple path in T_2 from u_1' to u_2' that contains the edges e_1' and e_2'. Since $u_1', u_2' \in W \subseteq U$, there is a path in $\mathcal{T}$ from u_1' to u_2'. This path also exists in T_2, since $e_2 \notin H$ is incident on U, while all Steiner tree edges incident on U are in H. Thus there are two distinct simple paths from u_1' to u_2' in T_2, contradicting the acyclicity of T_2.

Case 2: $e_1' \notin T$. This case is similar, except that we have to do a little more work to get to T_2. Since e_1' exists in T' but not in T, there is an edge $e_1 \in T$ such that (e_1, e_1') is a swap with respect to T. The lack of (Y, d)-inflating swaps for T implies that e_1 must be incident on U. Let $T_1 = T \setminus \{e_1\} \cup \{e_1'\}$. The edge e_2' can be swapped in for some edge $e_2 \ne e_1'$ in T_1. We next argue that e_2 is incident on U. Assume the contrary. Then (e_2, e_2') is a (Y, d)-inflating swap with respect to T_1. The algorithm's failure to find a (Y, d)-inflating swap implies that (e_2, e_2') is not a swap with respect to T. However, a simple structural argument then shows that (e_2, e_1') is a swap with respect to T. Moreover, it is (Y, d)-inflating, which contradicts the failure to find a (Y, d)-inflating swap. Thus, both e_1 and e_2 are incident on U and not internal to C_i. We conclude that C_i is internally connected in T_2, which leads to a contradiction by an argument analogous to Case 1.

The only thing that remains is to prove the quality of the witness $\mathcal{W}_{L,P}$. Since the components $C_1, \ldots, C_k$ are created by removing $S^L_{\le(d+1)}$ from T, we have $k \le (d+1)|S^L_{\le(d+1)}|$. Since the edges in H are in a Steiner tree and incident on $S^L_{\le(d+1)}$, there are at most $|S^L_{\le(d+1)}| - 1$ of them. Lemma 6 then implies the claim. □

4.3 The MstdbP Algorithm

Now we describe our MstdbP algorithm more completely. Recall that our MstdbP algorithm works in phases. Each phase employs algorithms MaxdmstP and MindmstP to improve either a high-degree vertex or a low-degree vertex in L; when both improvements fail, their failure is justified by two combinatorial witnesses. See Fig. 6 for a formal description.

```
Algorithm MstdbP($G, B_L, B_H, L, b$)

  Start with arbitrary minimum spanning tree $T$.
  repeat
     Compute $\delta$ so that $|S^L_{\le(B_L-\delta+1)}| \le b|S^L_{\le(B_L-\delta)}|$ and $|S_{\ge(B_H+\delta-1)}|$
     $\le b|S_{\ge(B_H+\delta)}|$.
     Call MaxdmstP with $d = B_H + \delta$ and $X = S^L_{\le(B_L-\delta+1)}$.
     Call MindmstP with $d = B_L - \delta$ and $Y = S_{\ge(B_H+\delta-1)}$.
  until both calls fail.
```

Fig. 6 Pseudo-code for MstdbP

Let us now look into a phase of the algorithm in more detail. Each phase of MstdbP begins by picking a δ such that

$$|S^L_{\le(B_L-\delta+1)}| \le b|S^L_{\le(B_L-\delta)}| \quad \text{and} \quad |S_{\ge(B_H+\delta-1)}| \le b|S_{\ge(B_H+\delta)}|,$$

where $b > 1$ is an input parameter. It is easy to show that one can always find such a δ in any range of size at least $2\log_b n$, and hence in particular, between $\max\{d_{\max}(T) - B_H, B_L - d^L_{\min}(T)\} - 2\log_b n$ and $\max\{d_{\max}(T) - B_H, B_L - d^L_{\min}(T)\}$. If $\delta < 0$, we are done and so we assume $\delta > 0$ for the rest of this section.

For the rest of the phase, vertices with degree at least $B_H + \delta$ are considered "high-degree" vertices and those in L with degree at most $B_L - \delta$ are considered "low-degree". We employ MaxdmstP and MindmstP to reduce the degree of a high-degree vertex and increase the degree of a low-degree vertex respectively. As alluded to earlier, we need to ensure that the improvements they perform do not interfere. For this purpose, we disallow the MaxdmstP algorithm from removing edges incident on $X = S^L_{\le(B_L-\delta+1)}$, and disallow the MindmstP algorithm from adding edges to $Y = S_{\ge(B_H+\delta-1)}$. Each subroutine either improves the potential significantly or returns a combinatorial witness. Each phase runs these two subroutines. The algorithm terminates if one of two things happen: the algorithm finds an MST with the required degree guarantees, or both subroutines output combinatorial witnesses on a particular tree T.

Lemma 8 guarantees that the algorithm terminates quickly. Theorem 9 shows that when both MaxdmstP and MindmstP fail with two combinatorial witnesses, at least one of the witnesses is good.

Lemma 8 *Algorithm* MstdbP *terminates in polynomial time.*

Proof Let T be the tree at the beginning of a particular phase and T' be the tree at the end of the phase. At the end of this phase, at least one of the following is true:

(i) T' has a potential function value at most $\phi(T) - 5^{\delta-1}$.
(ii) MaxdmstP and MindmstP both output combinatorial witnesses and the algorithm terminates.

The first step happens when one or both of MaxdmstP and MindmstP succeed; in this case Theorems 5 and 7 imply the claimed decrease in potential. Moreover, note that $\delta \ge \max\{d_{\max}(T) - B_H, B_L - d^L_{\min}(T)\} - 2\log_b n$ so that $d_{\max}(T) - B_H \le \delta + 2\log_b n$ and $B_L - d^L_{\min}(T) \le \delta + 2\log_b n$. Thus the total potential $\phi(T) \le n \cdot 5^{1+2\log_b n} 5^{\delta-1}$. Thus the decrease in potential is at least $\phi(T)/(n \cdot 5^{1+2\log_b n})$ in each phase, and the potential function decreases by half after $(n \cdot 5^{1+2\log_b n})$ phases. Since each node contributes at most $2 \cdot 5^n$ to the initial potential, the initial potential is at most exponential in n. The algorithm therefore terminates in polynomial time. □

Theorem 9 *Given a gragh G, a degree bound B and $b \in (1,2)$, there is a polynomial time algorithm that computes a tree T with cost at most OPT_B and degree at most $\frac{b}{2-b}B + O(\log_b n)$.*

Proof For a fixed B', suppose that we compute $c^{\lambda^{B'}}$ and set L to be set of nodes with a positive λ_u. Thus we have a fractional spanning tree τ that has maximum degree $d_{\max}(\tau)$ at most B' and $d^L_{\min}(\tau)$ at least B'. Executing MstdbP(G, B', B', b) on this cost function produces witnesses that certify that

$$B' \geq d_{\max}(\tau) \geq \frac{B'+\delta-3}{b} - 2\frac{|S^L_{\leq(B'-\delta+1)}|}{|S_{\geq(B'+\delta-1)}|} - 4$$

and

$$B' \leq d^L_{\min}(\tau) \leq b(B'-\delta+3) + \frac{|S_{\geq(B'+\delta-1)}|}{|S^L_{\leq(B'-\delta)}|} + 2.$$

By our choice of δ, at least one of $2b\frac{|S^L_{\leq(B'-\delta+1)}|}{|S_{\geq(B'+\delta-1)}|}$ and $\frac{|S_{\geq(B'+\delta-1)}|}{|S^L_{\leq(B'-\delta)}|}$ is less than $2b$; this is because the product of these two terms is at most $2b^2 < 4b^2$. In either case, this implies that $\delta \leq (b-1)B' + 5 + 6b$. However, $\delta \geq \max\{d_{\max}(T) - B', B' - d^L_{\min}(T)\} - 2\log_b n$, so that $d_{\max}(T) \leq bB' + 5 + 6b + 2\log_b n$ and $d^L_{\min}(T) \geq (2-b)B' - 5 - 6b - 2\log_b n$. For $1 < b < 2$ and $B = (2-b)B' - 5 - 6b - 2\log_b n$, this means that we have computed a tree with $d^L_{\min}(T) \geq B$ and $d_{\max}(T) \leq \frac{b}{2-b}B + \frac{2}{2-b}(5 + 6b + 2\log_b n)$. Using Lemma 2, we get a tree of cost at most $\text{OPT}_{LD(B)}$ and degree at most $\frac{b}{2-b}B + O(\log_b n)$. □

5 MSTs with Degree Bounds: A Quasipolynomial Time Algorithm

In this section, we describe our quasipolynomial time algorithm for the MSTDB problem. Given a graph $G = (V, E)$, a set of nodes L, a degree upper bound B_H and a degree lower bound B_L, our quasipolynomial time MstdbQ algorithm finds an MST T such that (a) every vertex has degree at most $B_H + O(\frac{\log n}{\log\log n})$ and (b) every vertex in L has degree at least $B_L - O(\frac{\log n}{\log\log n})$. If it fails, it produces a combinatorial witness to show that there exists no MST in the graph with degree at most B_H in which the vertices in L have degree at least B_L.

This algorithm uses the same framework as the algorithm in Sect. 4. Starting with an arbitrary MST, it proceeds in phases of improvement. Each phase is essentially a combination of the two algorithms MaxdmstQ and MindmstQ, which are described in Sects. 5.1 and 5.2. The MaxdmstQ algorithm attempts to decrease the degree of a node with high degree, and MindmstQ symmetrically attempts to increase the degree of a node in L which has low degree. If they both fail, the combinatorial witnesses they provide can be combined to produce a witness to the fact that there exists no MST in the graph with degree at most B_H in which the vertices in L have degree B_L or more.

To measure the progress made by our algorithm, we define a potential function $\phi_Q(T)$ on the set of all MSTs as follows:

$$\phi_Q(T) = \sum_v (2n)^{\deg(v)-B_H} + \sum_{v\in L}(2n)^{B_L-\deg(v)}.$$

While our algorithms MaxdmstQ and MindmstQ execute several swaps in order to improve the degree of one node, we shall nevertheless ensure that the potential decreases in each phase. Note that we have replaced the 5 in the definition of ϕ by $2n$; this is necessary to ensure the decrease in potential.

5.1 The MaxdmstQ Algorithm

Before describing our algorithm formally, we introduce some definitions and notation. Let T be the MST of G at the beginning of the current phase of the algorithm. Recall that $d_{\max}(T)$ denotes the maximum degree over all vertices in the current MST T.

Given an MST T, a set of nodes X, and an integer d, the aim of the MaxdmstQ algorithm is to reduce the degree of some vertex with degree d or more without reducing the degree of any vertex in X or increasing the degree of any vertex with degree at least $d-1$. This is achieved via a series of edge swaps. We shall show that for appropriately chosen X, the algorithm outputs a tree T' that has lower potential $\phi_Q(T')$ than T. If it fails to do this, it outputs a combinatorial witness proving a lower bound on maximum degree of any MST.

Throughout the MaxdmstQ algorithm, we maintain a center set $W \subset V$, and a partition $\mathcal{C} = \{C_1, \ldots, C_k\}$ of $V \setminus W$ into clusters. The initial center set W_0 is $S_{\geq(d-1)}$, and the initial clusters are the connected components created by deleting W_0 from T. In general, though, a cluster defined during the course of the MaxdmstQ algorithm is not necessarily internally connected. We also maintain a set R of restricted edges, that will not be allowed to be added to the tree; R is initially empty.

We use W to refer generally to the center set, and W_i to denote the center set at some specific iteration i of this phase. For a node $u \in W$, the clusters connected to u by tree edges are called the *children clusters* of u.

Let $\mathcal{T} \subseteq T$ be a Steiner tree on the nodes of W_0 (the Steiner nodes are the nodes in $V - W_0$). We say that we *freeze* the edges of $\mathcal{T}$ incident on W_0, meaning that the edges of $\mathcal{T}$ that are incident on W_0 are not allowed to be removed by any swap. As we show later, freezing these edges ensures that executing the edge swaps at the end of a phase of the algorithm results in a tree.

Unlike the MaxdmstP algorithm of Sect. 4.1, the MaxdmstQ algorithm does not restrict itself to making only (X, d)-deflating swaps. It instead finds a *sequence of swaps* that has a similar effect. More precisely, given a set X of nodes, a sequence of swaps $(e_1, e'_1), (e_2, e'_2), \ldots, (e_k, e'_k)$ is called *an (X, d)-deflating sequence* if (a) e_1 has exactly one endpoint in $S_{\geq d}$, (b) for all i, $1 \leq i < k$, e'_i and e_{i+1} have a common endpoint in $S_{\geq(d-1)}$, (c) for all i, $1 \leq i \leq k$, e_i is not incident on X, and (d) after all swaps in the sequence are performed, no node in $S_{\geq(d-1)}$ has larger degree than it does in T.

The aim of the MaxdmstQ algorithm is to construct an (X, d)-deflating sequence of swaps from swaps of a particular type. Given a set X of nodes, and a partition of all the nodes into a center set W and a set $\mathcal{C}$ of clusters, a swap (e, e') is called (X, W)*-deflating* if (a) e' is an inter-cluster edge, and (b) e has exactly one endpoint in W and is not incident on X. Recalling the definition of an (X, d)-deflating swap from Sect. 4.1, we note that an (X, W)-deflating swap is not necessar-

```
Algorithm MaxdmstQ($G, T, X, d$)

  Initialize $W = S_{\geq(d-1)}$, $R = \emptyset$. Let $\mathcal{C}$ be the components formed upon deleting
  $W$ from $T$.
  Let $\mathcal{T} \subseteq T$ be a Steiner tree on $S_{\geq(d-1)}$. Freeze the edges of $\mathcal{T}$ incident
  on $S_{\geq(d-1)}$.
  repeat
    Find an $(X, W)$-deflating swap $(e = (u, v), e' = (u', v'))$ such that $e$ is not
    frozen and $e' \notin R$.
    if no such swap exists
      break out of loop.
    Let $u$ be the endpoint of $e$ in $W$.
    Remove $u$ from $W$. Call $(e, e')$ the $u$-deflating swap.
    Merge: Form a new cluster $C_u$ by merging $u$ with the cluster containing $u'$,
        the cluster containing $v'$, and all the children clusters of $u$.
    Restrict: For each edge $(u, w)$ such that $w \in S_{\geq(d-1)} \setminus W$, add $(u, w)$ to $R$.
  until some node $u \in S_{\geq d}$ is removed from $W$.
  if removed a node $u \in S_{\geq d}$ from $W$
    then execute $(X, d)$-deflating sequence of swaps, starting with the $u$-deflating
        swap.
```

Fig. 7 Pseudo-code for MaxdmstQ

ily (X, d)-deflating, since e' may be incident on $S_{\geq(d-1)}$, or e itself may be incident on $W \setminus S_{\geq d}$.

The algorithm repeatedly finds (but does not perform) an (X, W)-deflating swap (e, e') such that e is not frozen and e' is not in R. After finding such a swap, it then removes the endpoint u of e that is in W from W and performs a *merge* step in which a new cluster C_u is formed by merging u with some other clusters. This swap (e, e') is called *the u-deflating swap*. The *restrict* step (see Fig. 7) of the algorithm prevents the swapping in of an edge between two nodes that initially have degree $(d-1)$ in T. The process is repeated until we either run out of (X, W)-deflating swaps that don't use frozen or restricted edges, or we remove a vertex u with degree at least d from W. In the former case, we construct a witness. In the latter case, we execute a particular (X, d)-deflating sequence of swaps, specified below, that decreases the degree of u. Figure 8 illustrates a run of the MaxdmstQ algorithm.

In order to specify the sequence of swaps executed at the end of the run of the MaxdmstQ algorithm, we first define, for any node u that is removed from W, *the u-deflating sequence* of swaps. It is defined inductively, as follows. Let (e, e') be the u-deflating swap, defined during the course of the MaxdmstQ algorithm. Note that because of the restrict step of the algorithm, at most one endpoint of e' is in the initial center set W_0. If neither of the endpoints of e' is in W_0, the u-deflating sequence is defined to contain just the u-deflating swap (e, e'). If one of the endpoints, say u', of e' is in W_0, the u-deflating sequence is defined inductively by adding the swap (e, e') to the beginning of the u'-deflating sequence. Since u' must be removed from the center set before u is, this sequence is well-defined.

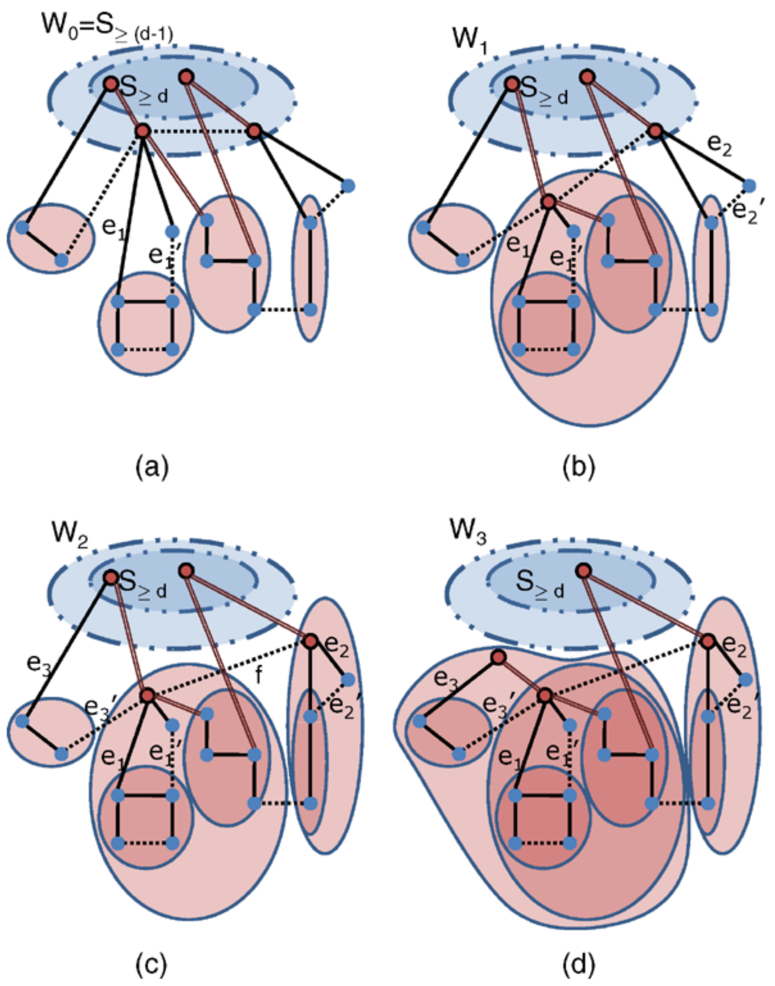

Fig. 8 A possible partial run of the MaxdmstQ algorithm. The frozen edges are shown by *double lines*. In each step, a swap is discovered that leads to the removal of a vertex from the center set W and the merging of some clusters. In step (**c**), the edge labeled f is added to R. Finally, in step (**d**), a vertex in $S_{\geq d}$ is removed from W. The execution (not shown) of the swaps (e_3, e_3') and (e_1, e_1') end this run

If u is the vertex of degree at least d removed from W, then the sequence of swaps executed in the last step of the algorithm is the u-deflating sequence. Lemmas 10 and 11 show that this sequence is in fact an (X, d)-deflating sequence. If the algorithm terminates instead by running out of (X, W)-deflating swaps, Lemma 12 and Theorem 14 show that we can interpret the remaining structure as a witness to the fact that the maximum degree of T is close to optimal.

For any node u removed from the center set, recall that C_u denotes the cluster formed by the merge step after u is removed (see Fig. 7 for a precise definition); note that since new clusters are formed by merging old clusters, a cluster C_u formed during one iteration lies wholly within some cluster in all subsequent iterations.

Lemma 10 *Suppose we remove a vertex u from the center set W_i in some iteration i. If we execute the u-deflating sequence, then* (a) *the degree of u decreases by one,* (b) *no node in $S_{\geq(d-1)}$ has larger degree than it does in T, and* (c) *the degree of no node changes by more than one.*

Proof We prove this by induction on the length of the u-deflating sequence. We shall strengthen the claim to add the condition that (d) the degree of every node outside C_u is unchanged.

In the base case, when the u-deflating sequence contains only one swap (e, e') with the endpoints of e' outside $S_{\geq(d-1)}$, the conditions (a), (b) and (c) follow immediately, and (d) follows from the merge step.

Now suppose that the claim holds for the u'-deflating sequence, which is augmented with the swap (e, e') to form the u-deflating sequence. The swap $(e = (u, v), e' = (u', v'))$ increases the degree of u' by one and the u'-deflating sequence decreases it by one; thus the degree of u' is unchanged. By construction v' is outside $S_{\geq(d-1)}$, and thus we have proved (a) and (b). The edge (u, v) does not lie on the path from u to u' in T, since if it did, it would be in $\mathcal{T}$ and hence frozen. Also, v is in the same cluster as v'; if it were not, the path in T from u to v' would contain another node from W_0, and thus (u, v) would be in $\mathcal{T}$ and hence frozen. Since the edge (u', v') is an inter-cluster edge, v' lies outside $C_{u'}$, and thus so does v. Part (d) of the inductive hypothesis then implies that the degrees of v and v' are unaffected by the u'-deflating sequence. Thus (c) follows. Finally the merge step implies that u, v, v', and $C_{u'}$ all lie within C_u, and thus (d) is true as well. □

We next show that executing the u-deflating sequence of swaps results in a tree. If executed in isolation, any single swap in the sequence obviously does not introduce a cycle. However, it is not as clear that doing them simultaneously does not disrupt the tree structure. In fact, the frozen edges of the Steiner tree play a crucial role here.

Lemma 11 *For any $u \in W_0$, the graph produced after performing the u-deflating sequence of swaps is a tree.*

Proof Let $(e_1, e'_1), (e_2, e'_2), \ldots, (e_k, e'_k)$ be the u-deflating sequence of swaps, so that e_1 is incident on u, and e'_i and e_{i+1} share an endpoint. Let T_j be the graph resulting after we execute the first j of these swaps, i.e. $T_0 = T$ and $T_j = T_{j-1} \setminus \{e_j\} \cup \{e'_j\}$. We show inductively that T_j is a tree.

The base case is immediate. Suppose that T_{j-1} is a tree for some j. It suffices to show that if $e_j = (u_j, v_j)$ could be replaced by $e'_j = (u'_j, v'_j)$ in T, it is a valid swap in T_{j-1} as well. In other words, if e_j lies on the tree path from v'_j to u'_j in T, then e_j lies on the tree path from v'_j to u'_j in T_{j-1} as well. Note that by construction e_j and e'_{j-1} share an endpoint u_j and that e'_{j-1} is the first edge in the sequence incident on C_{u_j}.

Consider the tree path in T from v'_j to u'_j (see Fig. 9). Note that by construction, all tree edges leaving C_{u_j} are incident on W_0. If the path lies wholly within C_{u_j}, then it must still exist in T_{j-1}, since all deleted edges $e_1, \ldots, e_{j-1}$ lie outside C_{u_j}. Otherwise this path can be decomposed into three segments P_1, P_2, and P_3, such

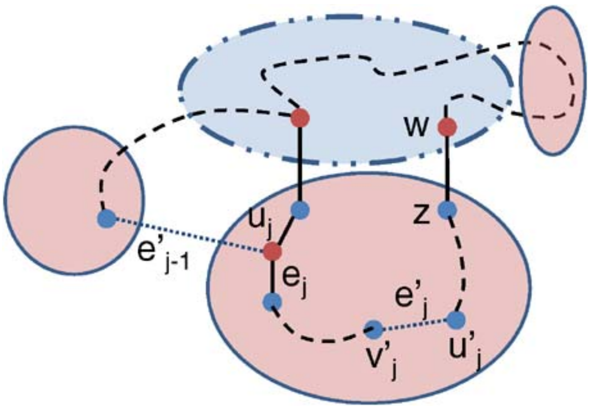

Fig. 9 Proof of Lemma 11. The *dotted edges* are not in T; the *dashed lines* indicate paths

that $v'_j \in P_1$, $u'_j \in P_3$, P_1 and P_3 are maximal paths contained in C_{u_j}, and P_2 is a path connecting P_1 and P_2. Without loss of generality, u_j (and thus e_j) lies in the segment P_1 containing v'_j. Let (w, z) be the last edge on P_2; i.e. $w \in W_0$ and z is connected to u'_j by P_3.

The paths P_1 and P_3 are preserved in T_{j-1}, since deleted edges $e_1, \ldots, e_{j-1}$ lie outside C_{u_j}. The path $P_2 \setminus (w, z)$ is in the Steiner tree $\mathcal{T}$, since it connects u_j to w in T; thus the freezing step ensures that it is preserved in T_{j-1}. Finally, the edge (w, z) must also exist in T_{j-1}: If $(w, z) \notin T_{j-1}$, then $T_{j-1} \setminus T$ must contain an edge e'_i, $1 \le i \le j-1$, incident on the component of $T \setminus \{(w, z)\}$ containing u'_j. Note that by construction, this component does not contain u_j, and therefore $e'_i \ne e'_{j-1}$. Since this component is contained in C_{u_j}, then e'_i is incident on C_{u_j}, which contradicts the observation that e'_{j-1} is the only edge incident on C_{u_j} in $T_{j-1} \setminus T$. Thus we have shown that the entire v'_j–u'_j path in T is preserved in T_{j-1}. The claim follows. □

5.1.1 The MaxdmstQ *Witness*

Now we show how to interpret the structure produced by running the MaxdmstQ algorithm as a high-degree witness, described in Sect. 4.1. Let W be the center set, $C_1, C_2, \ldots, C_k$ be the clusters, R the restricted edges, and T the tree when the algorithm terminates. Let T_X be the set of all edges in T in $X \times W$, and $F_{\mathcal{T}}$ be the set of Steiner tree edges that are frozen by the algorithm. We let $F = T_X \cup F_{\mathcal{T}}$. Let $\mathcal{W}_Q = (\{W, C_1, \ldots, C_k\}, W, F)$. We first argue that $\mathcal{W}_Q$ is a witness for the graph with the edges in R deleted.

Lemma 12 *$\mathcal{W}_Q = (\{W, C_1, \ldots, C_k\}, W, F)$ is a high-degree witness for $G' = (V, E \setminus R)$.*

Proof Suppose that there is some MST T' of G' containing F that does not contain an edge from C_i to W.

Let T_i be the set of edges in T connecting C_i to W. By construction, there are no inter-cluster tree edges, so T_i is non-empty. By the exchange property, there is a set $T'_i \subseteq T'$ such that $T_1 = T \setminus T_i \cup T'_i$ is an MST. Since C_i and $V \setminus C_i$ are disconnected in $T \setminus T_i$, there must be an edge $e' \in T_i'$ connecting them. By the exchange property, there is an edge $e \in T_i$ such that (e, e') is a swap with respect to T. Since $e' \in T'$, e' is not incident on W, and thus (e, e') is an (X, W)-deflating swap, contradicting our assumption. Hence the claim. □

Lemma 13 *At most* $2|W_0|$ *edges are frozen by the algorithm, where* W_0 *is our initial witness.*

Proof The average degree of a tree is at most two, and the degree of a Steiner vertex is at least two. Thus the average degree of the terminals in any Steiner tree is at most two. □

Finally, we show that when given an appropriate set X of vertices, the algorithm either computes a tree with lower potential, or produces a good witness.

Theorem 14 *Suppose we are given as input an MST* T, *an integer* d, *and a set* X *of nodes. Then the algorithm* MaxdmstQ, *when called with* $X = S^L_{\leq(B_L+B_H-d+1)}$ *runs in polynomial time, and either outputs a tree with potential function value at most* $\phi(T) - (2n)^{d-B_H-1}$, *or finds a witness that certifies that for any fractional MST* τ *of* G,

$$d_{\max}(\tau) \geq (d-2) - 5\frac{|S_{\geq(d-1)}|}{|S_{\geq d}|} - \frac{|X|}{|S_{\geq(d-1)}|}.$$

Proof Since there are at most $|V||E|$ swaps, each iteration runs in polynomial time. In each iteration, we remove a vertex from W, and thus the overall algorithm runs in polynomial time.

From Lemmas 10 and 11, we see that if the MaxdmstQ algorithm removes a vertex of degree d from the center set, it produces an MST T'. In this case, there is at least one vertex $u \in S_{\geq d}$ whose degree decreases by one, and no vertex has its degree raised above $d-1$. In addition, the degree of vertices in $S^L_{\leq(B_L+B_H-d+1)}$ does not decrease, and no vertex has its degree changed by more than one. Thus the decrease in potential due to u is at least $(2n-1)(2n)^{d-B_H-1}$, while the total increase due to the other vertices is at most $n(2n)^{d-B_H-1}$. The claimed decrease in potential follows.

If the algorithm is unable to improve any vertex of degree d, it instead halts after t iterations with another MST of degree d. Let $W_0 = S_{\geq(d-1)}$ be the initial center set, and let $W_t \supseteq S_{\geq d}$ be the final center set. The number of components obtained by deleting the edges incident to W_t is at least $(d-2)|W_t|$. The number of merges performed by the algorithm is at most $|W_0 \setminus W_t|$: To see this, note that every time we do a merge in the merge step, we remove a node from W. Therefore the number of clusters at the end of the algorithm is at least $(d-2)|W_t| - |W_0 \setminus W_t|$. According to Lemma 12, the witness $\mathcal{W}_Q$ is a high-degree witness for $G' = (V, E \setminus R)$. We can make use of the lower bound, however, by observing that all edges in R are incident on $W_0 \setminus W_t$, so that no MST of G can use more than $|W_0 \setminus W_t|$ of these edges. Using Lemma 12, Lemma 13, and the fact that the number of edges in T in $W_0 \times X$ is at most $|W_0| + |X|$, we conclude that for any fractional MST τ of G, $d_{\max}(\tau) \geq (d-2) - 5\frac{|S_{\geq(d-1)}|}{|S_{\geq d}|} - \frac{|X|}{|S_{\geq(d-1)}|}$. □

5.2 The MindmstQ Algorithm

Before describing our algorithm formally, we introduce some definitions and notation. Let T be the MST of G at the beginning of the algorithm. Recall that $d^L_{\min}(T) = \min_{v\in L} \deg_T(v)$ is the minimum degree over all nodes in L in tree T. Recall also that for a tree T, we denote by $S^L_{\leq d}$ the subset of nodes in L which have degree d or less in T.

The algorithm takes as input the graph G, a tree T, sets L and Y of vertices, and a positive integer d. The aim of the MindmstQ algorithm is to increase the degree of some node in $S^L_{\leq d}$ without increasing the degree of any vertex in Y or decreasing the degree of any vertex in $S^L_{\leq(d+1)}$. This is achieved via a series of edge swaps. We shall show that for appropriately chosen Y, the algorithm outputs a tree T' that has lower potential $\phi_Q(T')$ than that of T. If it fails to do this, it outputs a combinatorial witness proving an upper bound on the degree of L in any MST.

Throughout the MindmstQ algorithm, we maintain a center set W, and a partition $\mathcal{C} = \{C_1, \dots, C_k\}$ of $V \setminus W$ into clusters. The initial center set W_0 is $S^L_{\leq(d+1)}$, and the initial clusters are the connected components created by deleting W_0 from T. In general, during the course of the algorithm, a component may be split into several clusters, though each cluster remains internally connected in T. We use the term *intra-cluster edge* to refer to an edge whose endpoints are in the same cluster.

We also maintain a set N of special nodes and a set F of frozen edges. N is initially set to $S^L_{\leq(d+1)}$ and will be augmented during the algorithm. Let $\mathcal{T}$ be a Steiner tree on N such that $\mathcal{T} \subseteq T$ (with $V \setminus N$ as the Steiner nodes). We *freeze* the edges of $\mathcal{T}$ adjacent to N—that is, we disallow the algorithm from swapping out these edges. The set F is initialized to be the set of frozen edges of $\mathcal{T}$. As we show later, freezing these edges ensures that executing the sequence of edge swaps at the end of the algorithm results in a tree.

Given a set Y of nodes, a sequence of swaps $(e_1, e'_1), (e_2, e'_2), \dots, (e_k, e'_k)$ is called a (Y, d)-*inflating sequence* if (a) e'_1 has exactly one endpoint in $S^L_{\leq d}$, (b) for all i, $1 \leq i < k$, e_i and e'_{i+1} have a common endpoint in $S^L_{\leq(d+1)}$, (c) for all i, $1 \leq i \leq k$, e'_i is not incident on Y, and (d) after all swaps in the sequence are performed, no node in $S^L_{\leq(d+1)}$ has smaller degree than it does in T.

The aim of the MindmstQ algorithm is to construct a (Y, d)-inflating sequence of swaps from swaps of a particular kind. Given a set Y of nodes, and a partition of all nodes into a center set W and a set $\mathcal{C}$ of clusters, a swap (e, e') is called (Y, W)-*inflating* if (a) e is an intra-cluster edge, and (b) e' has exactly one endpoint in W and is not incident on Y. Recalling the definition of (Y, d)-inflating swaps in Sect. 4.2, we note that a (Y, W)-inflating swap (e, e') may not be (Y, d)-inflating, since e may be incident on $S^L_{\leq(d+1)}$ or e' may be incident on $W \setminus S^L_{\leq d}$.

The algorithm repeatedly finds (but does not execute) an (Y, W)-inflating swap (e, e') such that the edge e is not frozen. It then removes the endpoint $u' \in W$ of e' from W and performs a *merge* step to create a cluster $C_{u'}$ by merging u' with some other clusters (see Fig. 10). The swap (e, e') is called the u'-*inflating swap*. The following *split* step breaks the cluster containing e. Let u be the endpoint of e closer to u' in T. If u is not in W_0, we call such a swap a *basic swap*. If (e, e') is a basic swap, the *freeze* step adds u and v' to N and augments F appropriately (see Fig. 10). Note

```
Algorithm MindmstQ($G, T, L, Y, d$)

  Initialize $W = N = S^L_{\leq(d+1)}$. Let $\mathcal{C}$ be the components formed upon deleting $W$
   from $T$. Let $\mathcal{T} \subseteq T$ be the Steiner tree on $N$. Let $F$ be the edges of $\mathcal{T}$ incident
   on $N$.
  repeat
    Find a $(Y, W)$-inflating swap $(e = (u, v), e' = (u', v'))$ such that $e \notin F$.
    if no such swap exists
      break out of loop.
    Let $u'$ be the endpoint of $e'$ in $W$.
    Remove $u'$ from $W$. Call $(e, e')$ the $u'$-inflating swap.
    Merge: Form a new cluster $C_{u'}$ by merging $u'$ along with all the children
        clusters of $u'$, except for those which would involve a merge along an edge
        between $u'$ and $S^L_{\leq(d+1)}$.
    Split the cluster containing $(u, v)$ along $e$ into two clusters $C_u$ and $C_v$.
    Freeze: Let $u$ be the endpoint of $e$ closer to $u'$ in $T$. If $u \notin W_0$, add $u, v'$ to $N$.
        Augment $\mathcal{T}$ to a Steiner tree on $N$ and add the new edges of $\mathcal{T}$ incident
        on $N$ to $F$. Call the $u$–$v'$ path in $T$ the tail of this swap.
  until some node $u'$ in $S^L_{\leq d}$ is removed from $W$.
  if removed a node $u' \in S^L_{\leq d}$ from $W$
    then execute the $u'$-inflating sequence of swaps.
```

Fig. 10 Pseudo-code for MindmstQ

that for a non-basic swap, u is already in N. In either case, we call the u–v' path in T the *tail* of the swap. The process is repeated until we either run out of (Y, W)-inflating swaps that don't involve frozen edges or we remove a vertex $u' \in S^L_{\leq d}$ from W. In the former case, we construct a witness. In the latter case, we execute a particular (Y, d)-inflating sequence of swaps, specified below, that increases the degree of u' by one. As we shall see in Lemma 15, the merge, split, and freeze steps ensure that we never swap out more than one edge incident on any node, and that this sequence of swaps results in a tree. Figure 11 illustrates a run of the MindmstQ algorithm.

In order to specify the sequence of swaps executed at the end of the run of the MindmstQ algorithm, we first define, for any node u' that is removed from W, the *u'-inflating sequence* of swaps inductively as follows. Let (e, e') be the u'-inflating swap. We shall argue shortly that at most one endpoint of e is in the initial center set W_0. If neither of the endpoints of e is in W_0, then the u'-inflating sequence is defined to contain just the u'-inflating swap (e, e'). If one of the endpoints, say u, of e is in W_0, the u'-inflating sequence is defined inductively by adding the swap (e, e') to the beginning of the u-inflating sequence.

If u' is the vertex of degree at most d removed from W, then the sequence of swaps executed in the last step of the algorithm is the u'-inflating sequence. Lemma 15 shows that this sequence is in fact a (Y, d)-inflating sequence. Theorem 18 shows that if we run out of (Y, W)-inflating swaps, we can turn the resulting structure into a low-degree witness certifying an upper bound on the degree of L in any MST.

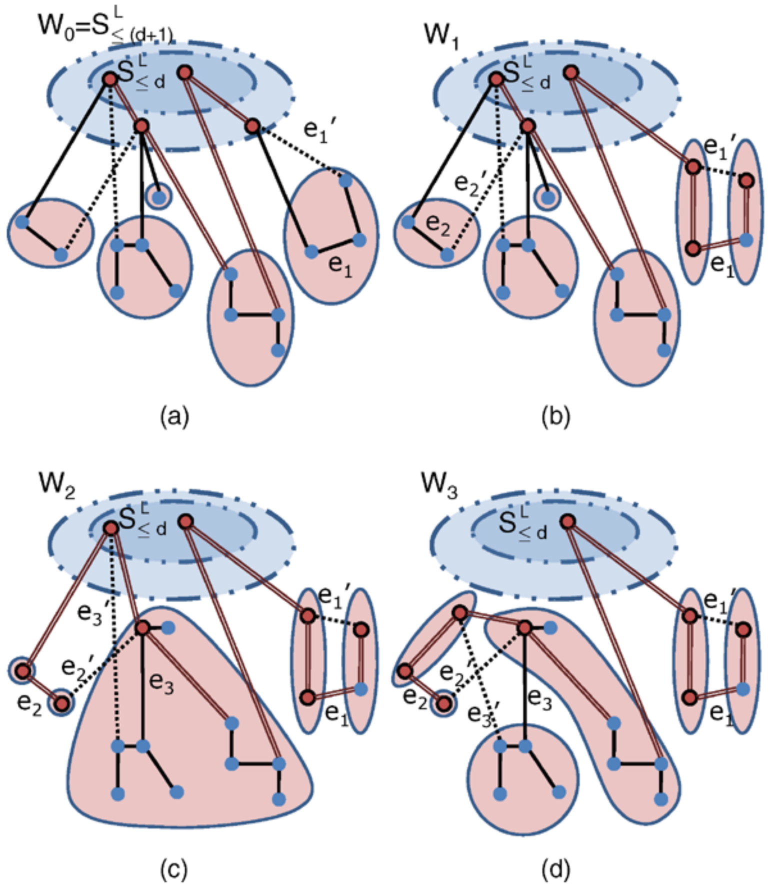

Fig. 11 A possible partial run of the MindmstQ algorithm. The frozen edges are shown by *double lines*, and the vertices in N are *circled*. In each step, a swap is discovered that leads to the removal of a vertex from the center set W and the merging and splitting of some clusters. In steps (**b**) and (**c**), new edges are also frozen. Finally, in step (**d**), a vertex in $S^L_{\le d}$ is removed from W. The execution (not shown) of the swaps (e_3, e'_3) and (e_2, e'_2) end this run

Lemma 15 *Suppose that we remove a vertex u' from W. Then if we execute the u'-inflating sequence of swaps defined above*, (a) *the degree of u' increases by* 1, (b) *no node in $S^L_{\le(d+1)}$ has smaller degree than it does in T*, (c) *the degree of no node changes by more than one*, *and* (d) *the resulting graph is a tree.*

Proof Let $(e_1, e'_1), (e_2, e'_2), \dots, (e_k, e'_k)$ be the u'-inflating sequence, where $(e_i, e'_i) = ((u_i, v_i), (u'_i, v'_i))$, $u' = u'_1$, and $u'_{i+1} \in S^L_{\le(d+1)}$ is the endpoint shared by e_i and e'_{i+1}.

We first observe that for any swap $((u_i, v_i), (u'_i, v'_i))$, u_i and v_i are not both in W_0. This is because at the time of discovery of this swap, (u_i, v_i) is an intra-cluster edge in T. The merge step of the algorithm ensures that we never merge clusters along an edge between two nodes in W_0. Thus an edge between two nodes in W_0 can never be an intra-cluster edge.

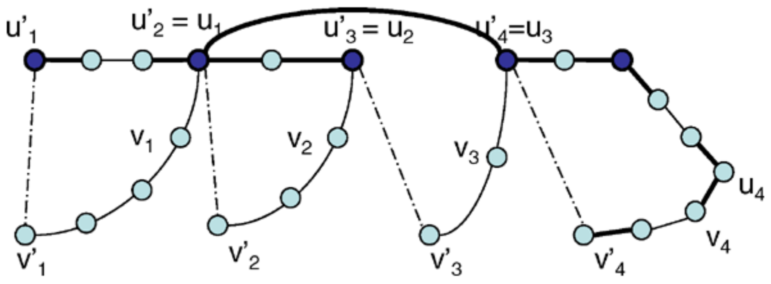

Fig. 12 The sequence of swaps. The *bold edges* are frozen. The *dark-colored nodes* are in W_0. The paths from u_i to v_i' are the tails

Recall that for swap (e_i, e_i'), we refer to the endpoint of e_i closer to u_i' as u_i. Thus e_i is on the u_i'–v_i path in T. If v_i were in N, then e_i would be frozen and we wouldn't have chosen the swap (e_i, e_i'). Thus for $i < k$, v_i must be different from u_{i+1}' (which is in $S^L_{\le(d+1)}$ and hence in N). Since e_i and e_{i+1}' share the endpoint u_{i+1}', we conclude that $u_i = u_{i+1}'$ for $i < k$. See Fig. 12.

For $i < k$, since $u_i = u_{i+1}'$ is in W_0, v_i is not in W_0. Moreover, (e_k, e_k') is a basic swap, so u_k and v_k are not in W_0. It follows that if u_k or v_i, for any i, is in L, then it has degree at least $d + 2$.

Let T_l be the tree resulting from executing the first j swaps, i.e. $T_0 = T$ and $T_l = T_{l-1} \setminus \{e_l\} \cup \{e_l'\}$. We show inductively that T_l is a tree. Let $N = N_t$ be the final set of special nodes. We strengthen the inductive claim to add the condition that the Steiner tree on N is undisturbed during the swaps.

Recall that the tail of swap (e_i, e_i') is defined to be the u_i–v_i' path in T. We make the following claims.

Claim 16 *The tail for swap i for $i < k$ intersects with the Steiner tree $\mathcal{T}$ on W_0 only in u_i.*

Proof Recall that $u_{i+1}' = u_i$ is in W_0. If the tail of the swap i contained a node $w \in \mathcal{T}$ different from u_i, the edge (u_i, v_i) would be on the u_i–w path and thus frozen. $\square$

Claim 17 *The tail for swap i is vertex-disjoint from the tail for swap j for $1 \le i < j \le k$.*

Proof First, let $i < j < k$. We observe that $(u_i, v_i) \neq (u_j, v_j)$. This holds because the splitting step of the algorithm ensures that (u_j, v_j) is an inter-cluster edge after the swap $((u_j, v_j), (u_j', v_j'))$ is discovered.

There is a Steiner tree path from u_i to u_j. Claim 16 implies that e_i and e_j are not on this path. If the tails of swaps i and j intersected in a node w, there is a simple path from u_i to u_j via w, that contains e_i and e_j and is therefore distinct from the Steiner tree path. This however contradicts the acyclicity of T.

Now let us look at the tail of swap k. Since this swap is basic, u_k and v_k are in N when swap i, $i \neq k$, is discovered. Suppose that the tail of swap k shares a vertex w with the tail of swap i, $i < k$. Then w has a path to u_i with e_i as its last edge. Since w

lies on the tail of swap k, the tail consists of the v'_k–w path and the u_k–w path. Either the u_i–u_k path or the u_i–v'_k path must go through w and thus contains e_i. Since v'_k, u_k and u_i are all in N, e_i must be frozen at the time of discovery of the swap (e_i, e'_i), which contradicts the fact that the swap (e_i, e'_i) was chosen. □

Let T_j be the tree resulting from executing the first j swaps, i.e. $T_0 = T$ and $T_j = T_{j-1} \setminus \{e_j\} \cup \{e'_j\}$. Let $N = N_t$ be the final set of special nodes.

We show inductively that T_j is a tree. The base case is immediate.

Consider the path from u'_j to v'_j in T; this path contains e_j by construction. It is enough to show that this path exists in T_{j-1}. This path consists of the u'_j–u_j path, along with the tail of swap j, which is the u_j–v'_j path. Since both u_j and u'_j lie in N, the path between them is not affected by the first $(j-1)$ swaps; indeed, each of the first $(j-1)$ swaps is a non-basic swap which deletes an edge incident on N, and the edges from $\mathcal{T}$ incident on N are frozen. Since the deleted edges $e_1, \ldots, e_{j-1}$ lie on the first $(j-1)$ tails which by Claim 17 are disjoint from the tail of swap j, the u_j–v'_j path in T is preserved in T_{j-1}. Thus the entire u'_j–v'_j path in T is preserved in T_{j-1}. It follows inductively that T_j is a tree, for $1 \le j \le k$, proving part (d) of Lemma 15.

We next show parts (a), (b), and (c) of Lemma 15. The tail of swap i contains u_i, v_i, and v'_i, and the tails of the swaps are vertex-disjoint according to Claim 17. This implies that the except for the equalities $u_i = u'_{i+1}$, the nodes involved in the swaps are all distinct. The node u'_1 gains an edge because of the swap (e_1, e'_1) and is not involved in any other swap, which proves part (a). The nodes u'_i, $i > 1$, are the only nodes involved in the swaps that lie in $S^L_{\le(d+1)}$. Each of these nodes gains and loses one unit of degree, implying part (b). Since the remaining involved nodes are all distinct from each other, part (c) follows. □

Theorem 18 *Suppose we are given as input an MST T, an integer d and a set Y of nodes. Then, the algorithm* MindmstQ *when called with $Y = S^L_{\le(B_L+B_H-d-1)}$ either outputs a tree with potential function value at most $\phi(T) - (2n)^{d-B_H-1}$, or finds a witness $\mathcal{W}$ that certifies that for any fractional MST τ of G,*

$$d^L_{\min}(\tau) \le (d+1) + 10\frac{|S^L_{\le(d+1)}|}{|S^L_{\le d}|} + \frac{|Y|}{|S^L_{\le d}|} - 6.$$

Proof Lemma 15 implies that if the algorithm removes a vertex of degree d from W, it can find a tree with the degree of this node increased by one. The calculation showing that the potential decreases is identical to that in Theorem 14.

If not, we show how to convert the structure produced by the MindmstQ algorithm into a low-degree witness of the type in Lemma 6. Suppose the algorithm begins with an MST T, a set Y, a degree d, and a set L, but fails to improve the degree of any vertex in L of degree at most d. It terminates after $t \ge 0$ iterations with W_t as the center set where $S^L_{\le d} \subseteq W_t \subseteq W_0 = S^L_{\le(d+1)}$, and $C_1, C_2, \ldots, C_{k'}$, as the clusters.

Let $W = W_t$ and $R = (W \times Y) \setminus T$. Let I be the set of inter-cluster edges in T; we set H to $F \cup I$. We shall argue that $\mathcal{W}_{L,Q} = (\{W, C_1, \ldots, C_{k'}\}, W, Y, R, H)$ is a low-degree witness.

Assume the contrary, i.e. $\mathcal{W}_{L,Q}$ is not a low-degree witness. Then there is an MST T' of G that contains H, excludes R and contains two edges $e'_1, e'_2 \notin H$ from cluster C_i to W.

The proof is similar to that of Theorem 7. Let (e_1, e'_1) be a swap that adds e'_1 to T, such that e_1 is not in H, and let T_1 be the tree $T \setminus \{e_1\} \cup \{e'_1\}$. Note that C_i is internally connected in T. If e_1 were not incident on W_0, (e_1, e'_1) would be (Y, W)-inflating, contradicting our assumption that the algorithm runs out of such swaps. Thus e_1 must be incident on W_0, and therefore C_i is internally connected in T_1 as well.

Now let (e_2, e'_2) be a swap that adds e'_2 to T_1 such that $e_2 \notin H$ and $e_2 \neq e'_1$. Let $T_2 = T_1 \setminus \{e_2\} \cup \{e'_2\}$. Suppose that e_2 is not incident on W_0. Then the swap (e_2, e'_2) is (Y, W)-inflating with respect to T_1. Since the algorithm runs out of (Y, W)-inflating swaps with respect to T, this swap does not exist in T. This implies that (e_2, e'_1) must be a swap in T. However, this swap is also (Y, W)-inflating, which is a contradiction. Thus e_2 is incident on W_0, and therefore C_i is internally connected in T_2.

Let u'_1 and u'_2 be the endpoints of e'_1 and e'_2, respectively, in W. Because C_i is internally connected in T_2, there is a path from u'_1 to u'_2 in T_2 containing the edges e'_1 and e'_2. There is a Steiner tree path in T from u'_1 to u'_2. This path is also present in T_2 since e_1 and e_2 are incident on W_0 and not frozen, while all Steiner tree edges incident on W_0 are frozen. Thus there are two distinct paths from u'_1 to u'_2 in T_2, which contradicts the acyclicity of T_2. We conclude that $\mathcal{W}_{L,Q}$ is a low-degree witness.

To finish the proof of Theorem 18, we compute the bound implied by Lemma 6. First we count the number of clusters created by the algorithm. The number of clusters is the number of components formed by deleting W_t from T, of which there are at most $(d+1)|W_t|$, plus the number of splits. There are a total of $|W_0 \setminus W_t|$ splits in the split step of the algorithm, each of which creates one additional cluster by splitting along the intra-cluster edge involved in the swap. There are also some clusters split (or rather, not merged) by tree edges adjacent to two $(d+1)$-degree nodes, but there can be at most $|W_0 \setminus W_t|$ of these. Thus, the number of clusters at the end of the algorithm is at most $(d+1)|W_t| + 2|W_0 \setminus W_t|$.

The set N has at most $W_0 + 2|W_0 \setminus W_t|$ nodes in it, and thus there are at most $2|W_0| + 4|W_0 \setminus W_t|$ edges in F (see proof of Lemma 13). Since each split (and each non-merge) contributes at most one inter-cluster edge, there are at most $2|W_0 \setminus W_t|$ edges in I. Therefore $|H| \leq 2|W_0| + 6|W_0 \setminus W_t|$.

Recall that Lemma 6 proves that for a witness $(\{W, C_1, \ldots, C_{k'}\}, W, Y, R, H)$ and any fractional tree τ,

$$d^L_{\min}(\tau) \leq \frac{k' + 2|W| + |U| + |Y| + |H| - 2}{|W|}.$$

Plugging in the values gives

$$d^L_{\min}(\tau) \leq \frac{(d+1)|W_t| + 2|W_0 \setminus W_t| + 2|W_t| + |W_t| + |Y| + 2|W_0| + 6|W_0 \setminus W_t| - 2}{|W_t|}.$$

Finally, noting that $W_0 = S^L_{\leq(d+1)}$ and $W_t \supseteq S^L_{\leq d}$, we obtain

$$d^L_{\min}(\tau) \leq (d+1) + 10\frac{|S^L_{\leq(d+1)}|}{|S^L_{\leq d}|} + \frac{|Y|}{|S^L_{\leq d}|} - 6. \qquad \square$$

5.3 The MstdbQ Algorithm

Our quasipolynomial MstdbQ algorithm is similar to the polynomial one, except that we use the better subroutines described above, and consequently use different parameters. Each phase employs algorithms MaxdmstQ and MindmstQ to improve either a high-degree vertex or a low-degree vertex in L; when both improvements fail, their failure is justified by two combinatorial witnesses.

Each phase of MstdbQ begins by picking a δ such that

$$|S^L_{\leq(B_L-\delta+1)}| \leq \frac{\log n}{\log\log n}|S^L_{\leq(B_L-\delta)}| \quad \text{and} \quad |S_{\geq(B_H+\delta-1)}| \leq \frac{\log n}{\log\log n}|S_{\geq(B_H+\delta)}|.$$

It is easy to show that one can always find such a δ in any range of length at least $2\frac{\log n}{\log\log n}$, and hence in particular, between $\max\{d_{\max}(T) - B_H, B_L - d^L_{\min}(T)\} - 2\frac{\log n}{\log\log n}$ and $\max\{d_{\max}(T) - B_H, B_L - d^L_{\min}(T)\}$. Without loss of generality, $\delta > 0$.

For the rest of the phase, vertices with degree at least $B_H + \delta$ are considered "high-degree" vertices and those in L with degree at most $B_L - \delta$ are considered "low-degree". We employ MaxdmstQ and MindmstQ to reduce the degree of a high-degree vertex and increase the degree of a low-degree vertex respectively. As alluded to earlier, we need to ensure that the improvements they perform do not interfere with each other. For this purpose, we disallow the MaxdmstQ algorithm from removing edges incident on $X = S^L_{\leq(B_L-\delta+1)}$, and disallow the MindmstQ algorithm from adding edges to $Y = S_{\geq(B_H+\delta-1)}$. Each subroutine either improves the potential significantly or returns a combinatorial witness. The algorithm terminates if one of the following happen: the algorithm finds an MST with the required degree guarantees, or both subroutines output combinatorial witnesses on a particular tree T. See Fig. 13 for a formal description of the algorithm.

```
Algorithm MstdbQ($G, L, B_L, B_H$)

  Start with arbitrary minimum spanning tree $T$.
  repeat
    Compute $\delta$ so that $|S^L_{\leq(B_L-\delta+1)}| \leq \frac{\log n}{\log\log n}|S^L_{\leq(B_L-\delta)}|$ and
    $|S_{\geq(B_H+\delta-1)}| \leq \frac{\log n}{\log\log n}|S_{\geq(B_H+\delta)}|$.
    Call MaxdmstP with $d = B_H + \delta$ and $X = S^L_{\leq(B_L-\delta+1)}$.
    Call MindmstP with $d = B_L - \delta$ and $Y = S_{\geq(B_H+\delta-1)}$.
  until both calls fail.
```

Fig. 13 Pseudo-code for MstdbQ

Lemma 19 guarantees that this is enough to make the algorithm terminate. Theorem 20 shows that when both MaxdmstQ and MINDMST fail with two combinatorial witnesses, at least one of the witnesses is good.

Lemma 19 *Algorithm* MstdbQ *terminates in quasi-polynomial time.*

Proof Let T be the tree at the beginning of a particular phase and T' be the tree at the end of the phase. At the end of this phase, at least one of the following is true:

(i) T' has a potential function value at most $\phi_Q(T) - (2n)^{\delta-1}$.
(ii) MaxdmstQ and MindmstQ both output combinatorial witnesses and the algorithm terminates.

The first step happens when one or both of MaxdmstQ and MindmstQ succeed; in this case Theorems 14 and 18 imply the claimed decrease in potential. Moreover, not that $\delta \geq \max\{d_{\max}(T) - B_H, B_L - d^L_{\min}(T)\} - 2\frac{\log n}{\log\log n}$ so that $\phi(T) \leq n \cdot (2n)^{1+2\frac{\log n}{\log\log n}}(2n)^{\delta-1}$. Thus the decrease in potential is at least $\phi_Q(T)/(n^{O(\frac{\log n}{\log\log n})})$ in each phase, and the potential function decreases by half after $(n^{O(\frac{\log n}{\log\log n})})$ phases. Since the initial potential is at most exponential in n, the algorithm terminates in quasi-polynomial time. □

Theorem 20 *Given a gragh G, and a degree bound B, there is a quasi-polynomial time algorithm that computes a tree T with cost at most OPT_B and degree at most $B + O(\frac{\log n}{\log\log n})$.*

Proof For a fixed B', suppose that we compute $c^{\lambda^{B'}}$ and set L to be set of nodes with a positive λ_u value. Thus we have a fractional spanning tree τ that has maximum degree $d_{\max}(\tau)$ at most B' and $d^L_{\min}(\tau)$ at least B'. Executing MstdbQ(G, B',B') on this cost function produces witnesses that certify that

$$B' \geq d_{\max}(\tau) \geq B' + \delta - O\left(\frac{\log n}{\log\log n}\right) - \frac{|S^L_{\leq(B'-\delta+1)}|}{|S_{\geq(B'+\delta-1)}|}$$

and

$$B' \leq d^L_{\min}(\tau) \leq B' - \delta + O\left(\frac{\log n}{\log\log n}\right) + \frac{|S_{\geq(B'+\delta-1)}|}{|S^L_{\leq(B'-\delta)}|}.$$

By our choice of δ, at least one of $2\frac{|S^L_{\leq(B'-\delta+1)}|}{|S_{\geq(B'+\delta-1)}|}$ and $\frac{|S_{\geq(B'+\delta-1)}|}{|S^L_{\leq(B'-\delta)}|}$ is less than $\frac{\log n}{\log\log n}$. This implies that $\delta \leq O(\frac{\log n}{\log\log n})$. However, $\delta \geq \max\{d_{\max}(T) - B', B' - d^L_{\min}(T)\} - 2\frac{\log n}{\log\log n}$, so that $d_{\max}(T) \leq B' + O(\frac{\log n}{\log\log n})$ and $d^L_{\min}(T) \geq B' - O(\frac{\log n}{\log\log n})$.

Choosing $B' = B + O(\frac{\log n}{\log\log n})$, this means that we have computed a tree with $d^L_{\min}(T) \geq B$ and $d_{\max}(T) \leq B + O(\frac{\log n}{\log\log n})$. Using Lemma 2, we get a tree of cost at most $\mathrm{OPT}_{LD(B)}$ and degree at most $B + O(\frac{\log n}{\log\log n})$. □

Acknowledgements We would like to thank the anonymous referees for several helpful comments.

Appendix: Tightness of Fischer's Analysis

In this section we show that Fischer's analysis of the local swap heuristic is nearly tight. Thus the previous techniques that did not consider multiswap improvements, i.e. *augmenting paths*, can provably not get a constant multiplicative factor.

Theorem 21 *Given any integer* $t > 1$, *there exists an asymptotic family of* (*unweighted*) *graphs* $\langle G_i \rangle$*and locally optimal trees* $\langle T_i \rangle$ *such that the maximum degree in* T_i *is at least* t *times the optimum for* G_i.

Proof We first show a family of instances showing a gap of two. We'll show an instance where a locally optimal tree has maximum degree $c \log n$ while the optimal solution has degree $c' \log n$ for some $c' \le \frac{c}{2}$. The vertex set consists of a root r, and sets L and R. The vertices in L are labelled by binary strings of length at most d and the vertices in R are labelled by tuples (x, j) where x is a binary string of length at most $(d-1)$ and j is a number between 1 and $(d - |x| - 1)$. It is easy to check that the total number of vertices is at most $(d+1)2^d$ which is $O(n)$ for $d = \log n, c < 1$. There is a white edges from r to each vertex in L. There are red edges between x and (x, j) for each j and blue edges between $(x, 2i)$ and $x0$ and between $(x, 2i + i)$ and $x1$. There are no L-L or R-R edges.

The locally optimal solution uses all white and red edges and thus the node corresponding to the empty string has degree d.[2] It is easy to check that this solution is locally optimal for the single swap heuristic. On the other hand, the solution using all white and blue edges has maximum degree (on vertex labelled 1) equal to $1 + \lceil \frac{d-1}{2} \rceil$. Further, a slight modification of this tree gives one with maximum degree at most $\frac{d}{2}$. Thus the local optimum has max degree at least twice the global one.

To get a gap of t, we use t-ary strings in the definitions of L and R and have blue edges from (x, j) to xi when $(j \bmod t = i)$. It is easy to check that for $d = c \log_t n, c < 1$, this construction gives the claimed gap. □

Moreover, note that since the above gap is shown for $\Delta_{\mathrm{OPT}} = O(\log n)$, the additive error of the one-swap heuristic is $\Omega(\log n)$.

References

1. Chan, T.M.: Euclidean bounded-degree spanning tree ratios. In: Proceedings of the Nineteenth Annual Symposium on Computational Geometry, pp. 11–19. ACM Press, New York (2003)
2. Chaudhuri, K., Rao, S., Riesenfeld, S., Talwar, K.: What would Edmonds do? Augmenting paths and witnesses for degree-bounded msts. In: Chekuri, C., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX-RANDOM. Lecture Notes in Computer Science, vol. 3624, pp. 26–39. Springer, New York (2005)

[2] Strictly speaking, r has the highest degree. However, by replacing the r–L edges by a binary tree with root r and L as the set of leaves, we can get an instance where it has degree 3.

3. Chaudhuri, K., Rao, S., Riesenfeld, S., Talwar, K.: A push-relabel algorithm for approximating degree bounded MSTs. In: ICALP (1). Lecture Notes in Computer Science, vol. 4051, pp. 191–201. Springer, New York (2006)
4. Edmonds, J.: Maximum matching and a polyhedron with 0–1 vertices. J. Res. Nat. Bur. Stand. **69B**, 125–130 (1965)
5. Edmonds, J.: Paths, trees, and flowers. Can. J. Math. **17**, 449–467 (1965)
6. Ellingham, M., Zha, X.: Toughness, trees and walks. J. Graph Theory **33**(3), 125–137 (2000)
7. Even, S., Tarjan, R.E.: Network flow and testing graph connectivity. SIAM J. Comput. **4**(4), 507–518 (1975)
8. Fischer, T.: Optimizing the degree of minimum weight spanning trees. Technical Report 14853, Dept. of Computer Science, Cornell University, Ithaca, NY (1993)
9. Fürer, M., Raghavachari, B.: Approximating the minimum-degree Steiner tree to within one of optimal. J. Algorithms **17**(3), 409–423 (1994)
10. Gavish, B.: Topological design of centralized computer networks–formulations and algorithms. Networks **12**, 355–377 (1982)
11. Goemans, M.X.: Minimum bounded degree spanning trees. In: FOCS, pp. 273–282. IEEE Computer Society, New York (2006)
12. Hoogeveen, J.A.: Analysis of Christofides' heuristic: some paths are more difficult than cycles. Oper. Res. Lett. **10**, 291–295 (1991)
13. Hopcroft, J., Karp, R.: An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. SIAM J. Comput. **2**, 225–231 (1973)
14. Jothi, R., Raghavachari, B.: Degree-bounded minimum spanning trees. In: Proc. 16th Canadian Conf. on Computational Geometry (CCCG), 2004
15. Khuller, S., Raghavachari, B., Young, N.: Low-degree spanning trees of small weight. SIAM J. Comput. **25**(2), 355–368 (1996)
16. Könemann, J., Ravi, R.: A matter of degree: improved approximation algorithms for degree-bounded minimum spanning trees. In: Proceedings of the Thirty Second Annual ACM Symposium on Theory of Computing: Portland, Oregon, May 21–23, 2000, pp. 537–546. ACM Press, New York (2000)
17. Könemann, J., Ravi, R.: Primal-dual meets local search: approximating MST's with nonuniform degree bounds. In: Proceedings of the Thirty-Fifth ACM Symposium on Theory of Computing, San Diego, CA, USA, June 9–11, 2003, pp. 389–395. ACM Press, New York (2003)
18. Krishnan, R., Raghavachari, B.: The directed minimum degree spanning tree problem. In: FSTTCS, pp. 232–243 (2001)
19. Papadimitriou, C.H., Vazirani, U.: On two geometric problems related to the traveling salesman problem. J. Algorithms **5**, 231–246 (1984)
20. Ravi, R., Singh, M.: Delegate and conquer: an LP-based approximation algorithm for minimum degree MSTs. In: ICALP (1). Lecture Notes in Computer Science, vol. 4051, pp. 169–180. Springer, New York (2006)
21. Ravi, R., Marathe, M.V., Ravi, S.S., Rosenkrantz, D.J., Hunt, H.B. III: Many birds with one stone: multi-objective approximation algorithms. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, pp. 438–447. ACM Press, New York (1993)
22. Ravi, R., Marathe, M.V., Ravi, S.S., Rosenkrantz, D.J., Hunt, H.B. III: Approximation algorithms for degree-constrained minimum-cost network-design problems. Algorithmica **31** (2001)
23. Singh, M., Lau, L.C.: Approximating minimum bounded degree spanning trees to within one of optimal. In: STOC '07: Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing, pp. 661–670. ACM Press, New York (2007)