Sublinear Approximations for Spanning Tree Congestion

Sunny Atalig
University of California at Riverside, USA

Marek Chrobak
University of California at Riverside, USA

Petr Kolman
Charles University, Faculty of Mathematics and Physics, Department of Applied Mathematics, Czechia

Hans Raj Tiwary
Charles University, Faculty of Mathematics and Physics, Department of Applied Mathematics, Czechia

Abstract

The spanning tree congestion of a connected graph G is the minimum, over all spanning trees T of G, of the maximum number of edges of G crossing a cut induced by deleting an edge of T. In the spanning tree congestion problem we are asked to compute a spanning tree with minimum congestion.

We describe two sublinear-ratio approximation algorithms for the spanning tree congestion problem (STC): a simple $\widetilde{O}(n^{8/9})$-approximation, and a more involved $\widetilde{O}(n^{2/3})$-approximation, where the notation $\widetilde{O}$ hides factors polynomial in $\log n$.. Until now, no algorithm with sublinear ratio was known.

We also prove that for any fixed $0 < \varepsilon < 1$, approximation of the spanning tree congestion within $2 - \varepsilon$ is NP-hard. This improves the previous best known lower bound 6/5.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Mathematics of computing → Graph algorithms; Theory of computation → Problems, reductions and completeness

Keywords and phrases spanning tree congestion, approximation algorithms, hereditary bisection, graph partitions, laminar decomposition, confluent flows

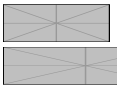
Digital Object Identifier 10.4230/LIPIcs...

Funding *Marek Chrobak*: Partially supported by NFS grant CCF-2153723.
Petr Kolman: Partially supported by GA ČR project 24-10306S.
Hans Raj Tiwary: Partially supported by the AGATE project funded from the Horizon Europe Programme under Grant Agreement No. 101183743.

1 Introduction

Let G be a connected graph and T a spanning tree of G. We can imagine routing every edge uv of G along the unique u–to–v path in T. The congestion $c(G, T)$ is the maximum number of these paths using one tree edge, and $\mathrm{STC}(G)$ is the minimum of $c(G, T)$ over all spanning trees T of G. Equivalently, the congestion of a tree edge e is the number of graph edges in the cut between the two subtrees of T resulting from removing e from T. The tree edge itself is counted.

Spanning tree congestion is difficult to approximate and no sublinear approximation algorithms for general graphs are known. Two useful approaches bound congestion in different ways. Chandran, Cheung, and Issac [2] construct a tree of congestion $O(\sqrt{mn}\log n)$ in polynomial time, where n and m are the numbers of vertices and edges. Combining their


© Sunny Atalig, Marek Chrobak, Petr Kolman, Hans Raj Tiwary;
licensed under Creative Commons License CC-BY 4.0

approach with the recent algorithm in [5] avoids the $\log n$-factor and yields a spanning tree of congestion at most $8\sqrt{mn}$ (see Lemma 2). Kolman [6] gives an $O(\Delta \log^{3/2} n)$-approximation, where Δ is the maximum degree, using hereditary bisection. The first bound is useful when the optimum is large; the second is useful when degrees are small.

In the connected min-max multiway cut problem, we are given a graph G with a set of t terminals $\Gamma \subseteq V(G)$. A connected multiway cut is a partition $V_1, \ldots, V_t$ of $V(G)$ such that for each $i \in [t]$: i) $G[V_i]$ is connected, and, ii) $|V_i \cap T| = 1$. The cost of such a cut is the maximum boundary size of any V_i. The goal is to find a connected multiway cut of minimum cost. This problem was studied by Kolman and Tiwary [7] who were motivated by its connection to the STC problem.

Our results.

We first give, in Section 2, a randomized $\widetilde{O}(n^{8/9})$-approximation for spanning tree congestion; this is the first algorithm with sublinear approximation ratio. One of the tools used in this algorithm is an approximation for the connected min–max multiway cut, and we provide an $\widetilde{O}(\sqrt{t})$-approximation for it in Appendix B, where t is the number of terminals.

Then, in Section 3, we describe our main result, which is an improvement of the simple algorithm.

▶ **Theorem 1.** *There is a randomized polynomial-time algorithm for spanning tree congestion with approximation ratio $\widetilde{O}(n^{2/3})$.*

On the hardness side, in Section 4 we prove that no polynomial-time $(2-\varepsilon)$-approximation exists for any fixed $0 < \varepsilon < 1$, unless $\mathsf{P} = \mathsf{NP}$.

Sketch of the algorithms.

Both approximation algorithms for spanning tree congestion use the following structure. Identify the high-degree vertices, called *hubs*. Separate them into connected regions containing one hub each. Within a region, remove its hub and partition the remaining vertices into connected parts with small boundaries. Construct a tree inside each part, join these trees through the hub, and finally join the regional trees. The algorithms differ in the two partitioning phases: how to obtain regions with only one hub, and how to partition the region with exactly one hub.

Our first algorithm combines two existing algorithms for STC — the spanning-tree construction of Chandran, Cheung, and Issac, which achieves congestion $\widetilde{O}(\sqrt{mn})$, and Kolman's $\widetilde{O}(\Delta)$-approximation algorithm — with a classical minimum-cut algorithm and our $\widetilde{O}(\sqrt{t})$-approximation of the connected min-max multiway partition with t terminals. Here, n, m, and Δ denote the number of vertices, the number of edges, and the maximum vertex degree of the input graph, respectively.

The second algorithm first constructs a new laminar decomposition of the input graph G, taking advantage of the bound on optimal congestion to control the boundary size of the sets in this decomposition, and the vertex degrees in each set. Then it recursively constructs spanning trees of graphs that are quotients of G with respect to the layers of this decomposition, eventually producing a low-congestion spanning tree of G.

In both algorithms, an important tool is a subroutine that constructs a spanning tree for a graph with one large degree vertex. To solve this task, the first algorithm uses an approximation of min-max connected partition combined with confluent flows. The second

algorithm uses confluent flows as well, but constructs a more sophisticated connected partition, with additional constraints on the distribution of edges between different parts.

The two key ingredients in our $\widetilde{O}(n^{2/3})$-approximation, namely the new laminar decomposition of a graph and the connected partitioning, might be of independent interest.

Preliminaries.

For a graph G, $V(G)$ (or V_G) denotes its vertex set and $E(G)$ (or E_G) its edge set. Throughout the approximation sections, G is a connected simple unweighted graph, $n = |V_G| \geq 2$, $m = |E_G|$, $k = \mathrm{STC}(G)$, and $L = \log n$. The degree of a vertex v in G is denoted $d_G(v)$. For disjoint vertex sets A, B in a graph G, let $E_G(A, B)$ be the set of edges between them and $e_G(A, B) = |E_G(A, B)|$. Write $G[A]$ for the subgraph induced by A. An auxiliary graph F may have positive integer edge weights that we denote by ω_F; for G, ω_G are unit weights. Parallel edges may be combined by adding their weights, and if we contract a subset of vertices we discard loops. Define

$$\delta_G(A) = E_G(A, V_G \setminus A), \qquad b_G(A) = \sum_{e \in \delta_G(A)} \omega_G(e).$$

The weighted degree is $d_G(v) = b_G(\{v\})$. For an edge set E', write $\omega_G(E')$ for its total weight, and for disjoint A, B write $\omega_G(A, B) = \omega_G(E_G(A, B))$.

Let $P_T(u, v)$ denote the unique u-to-v path in a spanning tree T of G. Define

$$c_{G,T}(e) = \sum_{\substack{uv \in E_G : \\ E(P_T(u,v)) \ni e}} \omega_G(uv), \qquad \ell_{G,T}(z) = \sum_{\substack{uv \in E_G : \\ V(P_T(u,v)) \ni z}} \omega_G(uv).$$

The first quantity is the *edge congestion* of edge e (also called its *load*) in a spanning tree T of a graph G; the second quantity is called *vertex load*. The congestion of a spanning tree T is $c(G, T) = \max_{e \in E(T)} c_{G,T}(e)$. A spanning tree of a graph on a single vertex has congestion zero.

We use the following consequence (see Appendix A.1) of the new algorithm in [5], which slightly improves the construction by Chandran, Cheung and Issac [2].

▶ **Lemma 2** ([2, 5]). *For every connected simple unweighted graph G on $n \geq 2$ vertices and m edges, a spanning tree of congestion at most $8\sqrt{mn}$ can be computed in polynomial time.*

We write T^* for an optimum spanning tree of G. Every graph edge uses at least one edge of T^*, whence

$$m \leq \sum_{e \in E(T^*)} c_{G,T^*}(e) \leq k(n-1) \leq kn. \tag{1.1}$$

Applying the bound (1.1) on the spanning tree from Lemma 2, we get that its approximation ratio is $O(\sqrt{mn}/k) = O(n/\sqrt{k})$. Whenever we refer to the ChandranCheungIssacTreetree below, we mean the construction of Lemma 2.

For $b > 1/2$, a b-balanced cut of G has both sides of size at most $b \cdot |V(G)|$. The *hereditary bisection* $\mathrm{hb}(G)$ is the maximum, over all subgraphs H with at least two vertices, of the minimum weight of a 2/3-balanced cut of H.

The following lemma roughly says that for a graph of hereditary bisection B, we can efficiently construct a spanning tree of congestion $\widetilde{O}(B)$.

▶ **Lemma 3** ([6]). *Let H be a connected positive-integer-weighted graph on n vertices, of total edge weight M_H and hereditary bisection B. There is an algorithm that in time polynomial in $n + M_H$ constructs a spanning tree with edge and vertex loads $O(B \log^{3/2} n)$.*

The proof (for completeness given in Appendix A) is a straightforward extension of the analysis of the hereditary-bisection based approximation algorithm for spanning tree congestion [6].

Min-max connected multiway cut.

Given a graph $G = (V, E)$ with positive integer weights and a set $\Gamma \subseteq V$ of t terminals, partition V into t parts such that each part is connected and contains exactly one terminal; the objective is to minimize the maximum weight of the edges leaving any part of the partition. The following approximation, proved in Appendix B, is used as a tool in the simple algorithm for spanning tree congestion.

▶ **Theorem 4.** *There exists a randomized $\widetilde{O}(\sqrt{t})$-approximation for the Min-max connected multiway cut with t terminals. Moreover, the maximum boundary of the constructed connected partition is $\widetilde{O}(\min\{\sqrt{t} \cdot OPT, \sqrt{M \cdot OPT}\})$ where M is the total edge weight in the graph.*

An important tool for proving Theorems 1 and 4 is confluent flow [3]. We will use the following result.

▶ **Theorem 5** (Theorem 20 [3]). *Given a splittable flow f on a graph with t sinks, there is a strongly polynomial time algorithm that finds a confluent flow satisfying the same demand and with node load at most $(1 + \ln t)$-times larger.*

2 A simple $\widetilde{O}(n^{8/9})$-approximation

For a degree threshold D, that we will set up later, define $H = \{v \in V : d_G(v) > D\}$ and $q = |H|$. Write $H = \{v_1, \ldots, v_q\}$ and call the vertices in H *hubs*.

The algorithm has three main phases: i) separate the hubs into connected regions, ii) split each region around its hub, and iii) construct trees in the resulting smaller parts, attach them to the hub in their region, and arbitrarily connect the subtrees into a spanning tree of G.

2.1 Tools

Let M be the size of a connected multiway cut separating the vertices of H; such a cut of size at most $(q-1) \cdot k$ can be constructed in polynomial time by performing $q-1$ minimum cuts between pairs of hubs, always within a current connected region containing at least two hubs; note that each individual mincut's size is at most k.

Let $R_1, \ldots, R_q$ be the corresponding connected subgraphs with $R_i \cap H = \{v_i\}$; then $b_G(R_i) \leq M$. We refer to these R_i's as *regions*.

▶ **Lemma 6.** *If $D \geq 4k$, then $q < 2n/D$ (so $q < n$).*

Proof. Since G is simple, each hub v_i has at most $|R_i| - 1$ neighbors inside R_i. An edge between regions contributes at most two to the sum of the hubs' outside degrees. Thus,

$$\sum_i d_G(v_i) \leq \sum_i (|R_i| - 1) + 2M \leq n - q + 2(q-1)k.$$

For $D \geq 4k$, we get $qD < \sum_i d_G(v_i) \leq n + 2qk \leq n + qD/2$, and hence $q < 2n/D$. ◀

For a region R_i, we define its *terminals* as the neighbors of v_i in R_i. We refer to induced subgraphs of R_i as *sub-regions*. If a sub-region contains exactly r terminals we call it an *r-terminal sub-region*. We want to partition $R_i \setminus \{v_i\}$ into connected 1-terminal sub-regions. Then the spanning trees of these sub-regions can be attached to v_i, each by its unique edge to v_i.

▶ **Lemma 7.** *For each $i \in [q]$, there is a partition of $R_i \setminus \{v_i\}$ into connected 1-terminal sub-regions, with $b_G(P) \le qk^2$ for every sub-region P in this partition.*

Proof. Fix $i \in [q]$ and write $R = R_i$, $v = v_i$, and note that $b_G(R) \le (q-1)k$. Every connected component of the forest $T^* \setminus \{v\}$ has boundary at most k; therefore it contains at most k neighbors of v (possibly none). Each connected component of $T^*[R \setminus \{v\}]$ thus contains at most k terminals. These connected components form our initial partition into sub-regions, that we now gradually modify.

First, for any r-terminal sub-region, with $r > 1$, delete from it $r-1$ edges of T^*, thus breaking it into connected 1-terminal sub-regions. It now remains to deal with 0-terminal sub-regions. We do it as follows. While there is a 0-terminal sub-region in the current partition, find one that has an edge connecting it to a 1-terminal sub-region (this is possible, by the connectivity of R) and merge it with this 1-terminal sub-region. This will eventually create a partition of $R \setminus \{v\}$ into connected 1-terminal sub-regions.

By construction, each 1-terminal sub-region P in this partition has (i) at most $k-1$ edges of $T^\star$ inside $R \setminus \{v\}$ to other partition's sub-regions, (ii) at most one edge of $T^\star$ to v, and (iii) at most $b_G(R)$ edges of $T^\star$ to $V(G) \setminus R$. As the spanning tree congestion of $T^\star$ is k, we get $b_G(P) \le k(k-1+1+b_G(R)) \le k(k+(q-1)k) = qk^2$. ◀

▶ **Corollary 8.** *There is a polynomial time algorithm that for each $i \in [q]$ constructs a partition of $R_i \setminus \{v_i\}$ into connected 1-terminal sub-regions, with $b_G(P) = \widetilde{O}(k\sqrt{mq})$ for every sub-region P in this partition.*

Proof. Lemma 7 bounds the boundaries of an optimal partition of R_i into connected 1-terminal sub-regions. To compute an approximate partition, we apply Theorem 4 separately to each connected component of $G[R_i \setminus v_i]$. Each resulting sub-region P has boundary $\widetilde{O}(k\sqrt{mq})$ within $G[R_i \setminus v_i]$, and it has exactly one edge to v_i, and at most $b_G(R_i)$ edges leaving R_i. Since $1 + b_G(R_i) \le qk \le k\sqrt{mq}$ for $q < n$ and $m \ge n-1$, we obtain $b_G(P) = \widetilde{O}(k\sqrt{mq})$. ◀

By the following lemma, for a connected subgraph with *bounded* boundary and *bounded* largest degree, we can construct a spanning tree with bounded congestion.

▶ **Lemma 9.** *Let $P \subseteq V(G)$ be nonempty with $G[P]$ connected, and put $D_P = \max_{u \in P} d_G(u)$. Then Kolman's algorithm [6] constructs a spanning tree T_P of $G[P]$ with*

$$c(G[P], T_P) = O\Big(k \max\{D_P, b_G(P)\} \log^{3/2} n\Big). \tag{2.1}$$

Proof. We start by showing that $\mathrm{hb}(G[P]) \le k \max\{D_P, b_G(P)\}$. Take any $X \subseteq P$ with $|X| \ge 2$. Mark its vertices in T^* and choose a weighted centroid z, so that every component of $T^* - z$ contains at most $|X|/2$ marks. Group these components, and the singleton $\{z\}$ if marked, into two sides each containing between $|X|/3$ and $2|X|/3$ marks. To do so, take a block of size at least $|X|/3$ if one exists; otherwise accumulate blocks until this threshold is first reached.

Every edge of $G[X]$ crossing this cut has its tree path pass through z. If $z \in P$, that path uses an edge of T^* incident with z; there are at most D_P such edges. If $z \notin P$, the path

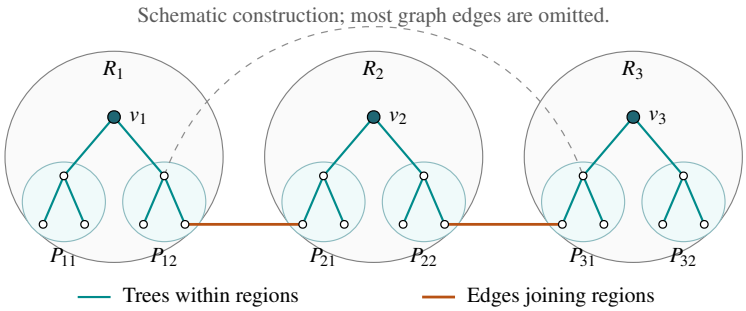


Figure 1 The simpler algorithm. Each region R_i contains one hub v_i. The connected parts P_{ij} each contain one neighbor of v_i; their internal trees are joined to v_i through these neighbors. A tree of the region quotient then joins the regional trees. Solid edges show the constructed tree; the dashed edge represents another graph edge. Most graph edges are omitted.

uses an edge of T^* leaving P; there are at most $b_G(P)$ such edges. Their loads are at most k, giving the claimed balanced cut bound. Deleting edges cannot increase the minimum balanced-cut cost, so the bound holds for all subgraphs of $G[P]$. The tree bound follows from the hereditary-bisection black box, Lemma 3. ◀

2.2 Algorithm and analysis

Write HereditarySpanTree for the construction of Lemma 9, and ChandranCheungIssacTree for the construction of Lemma 2. Figure 1 illustrates the construction of the following algorithm.

Algorithm 1 SimpleSpanTree(G)

$n \leftarrow |V(G)|$, $D \leftarrow 4n^{8/9}$
$H \leftarrow \{v \in V(G) : d_G(v) > D\}$, $q \leftarrow |H|$
if $q = 0$ **then**
  return HereditarySpanTree(G) ▷ Lemma 3
$T_0 \leftarrow$ ChandranCheungIssacTree(G) ▷ Lemma 2
if $q = n$ **then**
  return T_0
Write $H = \{v_1, \ldots, v_q\}$
$R_1, \ldots, R_q \leftarrow$ parts of connected multiway cut for (G, H) ▷ $q-1$ successive mincuts
for $i = 1, \ldots, q$ **do**
  for all connected components C of $G[R_i \setminus \{v_i\}]$ **do**
    if C contains at least two terminals **then** ▷ terminals = neighbors of v_i
      Partition C into connected 1-terminal sub-regions using Theorem 4
  for all sub-regions P of R_i **do**
    $T'_P \leftarrow$ HereditarySpanTree($G[P]$) ▷ Lemma 9
  $T_i \leftarrow$ join each T'_P to v_i through its unique terminal ▷ spanning tree of R_i
$T \leftarrow$ join $T_1, \ldots, T_q$ into a tree arbitrarily ▷ spanning tree of G
return the better of T and T_0

▶ **Theorem 10.** *The approximation ratio of the* SimpleSpanTree *algorithm is* $\widetilde{O}(n^{8/9})$.

Proof. Suppose first that $1 \le q < n$. Every connected 1-terminal sub-region P from step (13) has, by construction, largest degree $D_P \le D$, and, by Corollary 8, boundary $b_G(P) = \widetilde{O}(k\sqrt{mq})$. By Lemma 9, its spanning tree T'_P from step (15) has congestion $\widetilde{O}(k(D + k\sqrt{mq}))$. When T_P is joined to its hub, edges with an endpoint outside P add at most $b_G(P)$ to the congestion of any edge in T_P. The edge joining T_P to its hub v_i has congestion at most $b_G(P)$ in T_i. When the trees T_i are joined, only the M edges between the regions can increase congestion of edges in these trees, and each edge joining these trees has load at most M. Thus, if $D \ge 4k$, the spanning tree T has congestion at most

$$k\,\widetilde{O}(D + k\sqrt{mq}) + \widetilde{O}(k\sqrt{mq}) + M = k\,\widetilde{O}\Big(D + nk^{3/2}/\sqrt{D}\Big)\,, \tag{2.2}$$

where the bounds $q < 2n/D$ (Lemma 6), $M \le (q-1)k$, and $m \le k(n-1)$ (relation (1.1)) were used; the approximation ratio is $\widetilde{O}\Big(D + nk^{3/2}/\sqrt{D}\Big)$. If $k \le n^{2/9}$, the choice $D = 4n^{8/9}$ ensures that the approximation ratio is $\widetilde{O}(n^{8/9})$.

Recall that the Chandran–Cheung–Issac construction of Lemma 2 has approximation ratio $O(n/\sqrt{k})$; thus, if $k > n^{2/9}$, this ratio is $O(n^{8/9})$.

The case $q = 0$ has ratio $O(D\log^{3/2} n)$ by Lemma 3. If $q = n$, the congestion of any edge adjacent to a leaf is at least $d_G(u) > D$, so $k \ge d_G(u) > D$, and again the Chandran–Cheung–Issac bound suffices. ◀

The preceding proof identifies two distinct losses of the simple algorithm.

Separating the hubs. The first partition has total cut size at most $(q-1)k$, and even some individual region may have boundary of that order. This leads to the qk^2 bound in Lemma 7. In contrast, our improved algorithm will find small-boundary regions progressively in several rounds: in each round, it constructs regions with boundary $O(k)$ for at least half of the hubs, then contracts these regions into single vertices, and repeats the same step on the modified graph. This way, after at most logarithmically many rounds, it separates all hubs.

Controlling the smaller parts. A connected min–max multiway cut controls $b_G(P)$ but gives no further guarantee about the edges inside P. The internal-tree bound therefore pays $k\max\{D, b_G(P)\}$. The main algorithm instead constructs a partition $\mathcal{P}$ that also controls, for every small-boundary set C, the edges of $G[C]$ between different parts of $\mathcal{P}$. This simultaneous guarantee is what permits efficient routing between parts while Kolman's algorithm constructs their internal trees.

3 An $\widetilde{O}(n^{2/3})$ Approximation Via a Laminar Decomposition

We first describe the construction for a degree threshold $D \ge 4k$; the algorithm will try powers of two, without knowing k. It tries $D = 2^i$ for $2 \le i \le \lceil \log_2(8m) \rceil$, discards failed guesses, and retains the least-congested tree obtained.

For disjoint vertex sets A, B in G, let $\mathrm{mincut}_G(A,B) = \min\{b_G(X) : A \subseteq X \subseteq V_G \setminus B\}$.

As in Section 2, put $H = \{v : d_G(v) > D\}$ and call its vertices *hubs*. The algorithm has two phases. First, it constructs a laminar decomposition of G which is a laminar family $\mathcal{L}$ of connected sets that separate the hubs; each set in the family has boundary $O(D)$. Next, the algorithm processes the laminar tree of $\mathcal{L}$ and gradually constructs a spanning tree.

Graph Quotients.

Given a graph G, a *bag* is a connected set of its vertices. For a partition $\mathcal{P}$ of V_G, the *quotient* of G by $\mathcal{P}$ is the graph $F = G/\mathcal{P}$ that has vertex set $\mathcal{P}$ and where distinct sets $A, B \in \mathcal{P}$

are adjacent if and only if some vertex in A is adjacent in G to some vertex in B; the edge AB has weight $\omega_F(AB) = e_G(A, B)$. For a weighted input graph, this weight is instead the sum of the weights of its edges between A and B; this convention applies when contracting a quotient again.

For a collection $\mathcal{S}$ of disjoint sets of G, we write $G/\mathcal{S}$ for $G/\mathcal{S}'$ where $\mathcal{S}' = \mathcal{S} \cup \{\{v\} : v \in V_G \setminus \bigcup_{S\in\mathcal{S}} S\}$, that is, we view the disjoint collection $\mathcal{S}$ as a representation of the partition of V_G that contains the sets in $\mathcal{S}$ and in which all vertices not contained in any bag in $\mathcal{S}$ appear as size-one sets. For notational simplicity, in quotient graphs, we often identify each singleton bag $\{v\}$ with v. We say that $G/\mathcal{S}$ is a quotient of G *at level* 1, and if F is a quotient of G at level $i \geq 1$ and $\mathcal{S}'$ is a collection of disjoint sets of F, then $F/\mathcal{S}'$ is a quotient of G *at level* $i + 1$.

One can naturally extend the notion of a quotient for a laminar family $\mathcal{L}$ as follows. Recall that a family $\mathcal{L}$ is *laminar* if any two of its members are disjoint or one contains the other; the depth of $\mathcal{L}$ is the maximum number of members in an inclusion chain. We denote the set of inclusion-wise maximal members of $\mathcal{L}$ by $\mathcal{L}_1$ and – if it is unique – call it the *root* of $\mathcal{L}$. By $\mathcal{L}_2$ we denote the members that are inclusion-wise maximal in $\mathcal{L} \setminus \mathcal{L}_1$. If $\mathcal{L}_1$ is the root then $\mathcal{L}_2$ are the maximal sets in $\mathcal{L}$ that are properly contained in $\mathcal{L}_1$. We will view $\mathcal{L}_2$ as a collection of disjoint subsets – either of $\mathcal{L}_1$ or the entire vertex set depending on the context – and denote by $G/\mathcal{L}$ the *quotient* $G/\mathcal{L}_2$.

3.1 Construction of the Laminar Graph Decomposition

Let G be a graph. For $S \subseteq V_F$, where F is a quotient of G, let $S^\uparrow = \bigcup_{A\in S} A$ denote the union of the sets corresponding to the vertices in S. For $S \subseteq V_F$, if F is a quotient of G at level 1, then let $S^* = S^\uparrow$, and if F is a quotient of G at level $i \geq 2$, then let $S^* = \bigcup_{A\in S} A^*$. For a collection $\mathcal{C}$ of subsets of V_F, define $\mathcal{C}^\uparrow = \{S^\uparrow : S \in \mathcal{C}\}$. Informally speaking, we will construct a hierarchy of quotient graphs and for $S \subseteq V_F$ for some quotient F in this hierarchy every vertex $v \in V_F$ represents a subset of vertices in the parent graph that were contracted. $S^\uparrow$ then are the vertices that are one level up and that were contracted to get S while S^* are the vertices in the original graph. For example, suppose we start with G. Let F be a quotient of G and let H be a quotient of F. Then, for $S \subseteq V_H$, $S^\uparrow$ are vertices in V_F that were contracted to get S whereas S^* are the vertices in V_G that were eventually contracted to get S.

Given a weighted graph G and a subset $H \subset V_G$ of size at least 2, the *minimum isolating cuts* for H is a collection of disjoint sets $\{S_h : h \in H\}$ satisfying for each $h \in H$, $S_h \cap H = \{h\}$ and $b_G(S_h) = \text{mincut}_G(\{h\}, H \setminus \{h\})$. Minimum isolating cuts for H, in which each set S_h induces a connected subgraph, can be constructed in polynomial time by repeated calls to a max-flow algorithm (e.g., [8]); in the following, we refer to this algorithm as the IsolatingSets. Its input is the pair (G, H), and the output is a collection of the minimum isolating cuts for H.

Throughout the laminar construction, contracting an isolating set S_h retains the name h for the resulting vertex; all other vertices keep their names. We use the same names in the local graphs $G[R]/\mathcal{L}$ that are used in the construction. Their vertex sets are therefore subsets of V_G.

In our laminar family, the boundary of each non-root member set will be a minimum cut in G that separates (at the time when this set is being added to the family) one yet non-contracted hub from all other yet non-contracted hubs. The boundary of the root V_G is empty. The following lemma follows directly from sub-modularity of cuts (cf. Lemma 3.1 in the paper [8]); for the sake of completeness, we give its proof (in Appendix).

▶ **Lemma 11.** *Let $H \subseteq V$, $H' \subseteq H$, and let $\mathcal{S} = \{S_h : h \in H'\}$ be pairwise disjoint sets satisfying, for each $h \in H'$, $S_h \cap H = \{h\}$ and $b_G(S_h) = \text{mincut}_G(\{h\}, H \setminus \{h\})$. Then, for every pair of disjoint sets $A, B \subseteq H \setminus H'$,*

$$\text{mincut}_{G/\mathcal{S}}(A, B) = \text{mincut}_G(A, B). \tag{3.1}$$

More generally, if we identify the contracted vertex S_h with the corresponding vertex h and set $R = (V \setminus \bigcup_{h \in H'} S_h) \cup H'$, then (3.1) holds for all disjoint $A, B \subseteq R$.

Proof. Identify each contracted vertex S_h with h and put $R = \left(V \setminus \bigcup_{h \in H'} S_h\right) \cup H'$. We prove a slightly more general statement, namely that equality (3.1) holds for all disjoint $A, B \subseteq R$.

For disjoint sets $A, B \subseteq R$, fix a minimum A–B cut X in G, that is, $A \subseteq X \subseteq V \setminus B$ minimizing $b_G(X)$. We are going to construct an A–B cut in $G/\mathcal{S}$ of the same size. Process the sets S_h from $\mathcal{S}$ one at a time. If $h \in X$, replace X by $X \cup S_h$; note that the set $X \cap S_h$ separates h from $H \setminus \{h\}$. Thus, by the optimality of S_h, and by sub-modularity of cuts we get

$$b_G(X \cup S_h) + b_G(X \cap S_h) \leq b_G(X) + b_G(S_h) \leq b_G(X) + b_G(X \cap S_h).$$

Hence this replacement does not increase the boundary. If $h \notin X$, apply the same argument to $V \setminus X$ and use symmetry of the cut function: replacing X by $X \setminus S_h$ also does not increase its boundary.

Each replacement puts the entire set S_h on the side of X containing h. Since $S_h \cap R = \{h\}$, no terminal of A or B changes sides, and X remains an A–B cut. Disjointness of the sets S_h ensures that subsequent replacements do not split a set already processed. For the resulting cut X, $X/\mathcal{S}$ is therefore a cut of $G/\mathcal{S}$ of weight at most $\text{mincut}_G(A, B)$. Conversely, every cut of the quotient induces a cut of G with exactly the same weight and the same terminal separation. ◀

Algorithm 2 LaminarConstruction(F, D)

```
H_F ← {v ∈ V_F : d_F(v) > D}
if |H_F| ≤ 1 then
    return {V_F}                          ▷ backtracking will convert {V_F} to V_G
C ← IsolatingSets(F, H_F)
C_1 ← {S ∈ C : b_F(S) ≤ D}
if C_1 = ∅ then
    return Fail                           ▷ unsuccessful guess of D
F_1 ← F/C_1
C_2 ← LaminarConstruction(F_1, D)
if C_2 = Fail then
    return Fail
return C_1 ∪ C_2^↑
```

▶ **Lemma 12** (Laminar Graph Decomposition)**.** *For every $D \geq 4k$, the algorithm* LaminarConstruction *applied on the graph G constructs a laminar family $\mathcal{L}$ with the following properties:*

1. *The depth of $\mathcal{L}$ is $O(\log(|H| + 2))$, where $H = \{v \in V : d_G(v) > D\}$.*

2. *For each* $S \in \mathcal{L}$*, a)* $G[S]$ *is connected, and, b)* $b_G(S) \leq D$*.*

3. *For each* $S \in \mathcal{L}$*,* $\left|\left(S \setminus \bigcup_{A \in \mathcal{L}, A \subsetneq S} A\right) \cap H\right| \leq 1$*.*

4. $V_G \in \mathcal{L}$*.*

Proof. Each contracted set is connected and has boundary at most D, so the weighted degree of the new vertex corresponding to it is at most D. Contractions of other bags preserve this degree. Original singleton vertices also retain their original degrees. Hence, H_F consists exactly of the original hubs not yet contracted, and $H_F \subseteq H$.

Consider a recursive call LaminarConstruction(F, D), and let $q = |H_F| \geq 2$. Any two hubs in H_F can be separated in G by a fundamental cut of an optimum spanning tree, of cost at most k. More strongly, by deleting $q-1$ edges in the optimum tree, we can separate the q hubs into disjoint sets $\{S_h^*\}_{h \in H_F}$ where $\sum_{h \in H_F} b_G(S_h^*) \leq 2(q-1)k$. Lemma 11 preserves these cut values in F. Consequently, $\sum_{h \in H_F} b_F(S_h) \leq 2(q-1)k \leq qD/2$, implying that fewer than $q/2$ isolating sets for H_F have boundary greater than D. It follows that for at least half of the hubs from H_F, the connected subsets S_h get contracted, and the number of hubs decreases by a factor of two with each recursion level, implying that the depth of $\mathcal{L}$ is $O(\log(|H|+2))$. This proves Property 1.

Sets contracted in one round are disjoint, and a later contracted bag either contains any earlier bag or is disjoint with it. This proves laminarity. Connectedness is preserved under lifting connected sets of connected bags. The identity $b_G(S^*) = b_F(S)$ gives Property 2b for contracted sets, and $b_G(V_G) = 0$ gives it for the root.

By construction, at every level of recursion, every set $S \in \mathcal{C}$ from the collection of minimum isolation cuts for H_F in F, contains exactly one hub from H_F. This implies Property 3.

The single-set family $\{V_F\}$ returned in Line 3 will be gradually expanded (in line 12) in the backtracking phase of the recursion and eventually converted to set V_G, proving Property 4. ◀

3.2 Construction of the Spanning Tree

Let $\mathcal{L}$ be the laminar family constructed by LaminarConstruction(G, D), and for $S \in \mathcal{L}$, let $\mathcal{L}(S)$ be the set of *all* subsets of S that belong to $\mathcal{L}$, including S. We assume that two algorithms are available: HereditarySpanTree, the approximation algorithm based on hereditary bisection [6] (see Lemma 3), and UniqueHighDegreeSpanTree, the algorithm of Lemma 17. The latter receives the graph F, the original high-degree vertex h, and the parameter D; the hereditary routine receives the graph F with all degrees at most D. By $G[\mathcal{L}]$ we denote the subgraph of G induced by all the elements in the sets of the laminar family $\mathcal{L}$.

Algorithm 3 LaminarSpanTree($\mathcal{L}, D$)

if depth($\mathcal{L}$) = 1 **then**
  $F \leftarrow G[\mathcal{L}]$
else
  $F \leftarrow G[\mathcal{L}]/\mathcal{L}$
if there is $h \in V_F$ with $d_F(h) > D$ **then**
  $T_1 \leftarrow$ UniqueHighDegreeSpanTree(F, h, D)
else
  $T_1 \leftarrow$ HereditarySpanTree(F)
if depth($\mathcal{L}$) = 1 **then**
  return T_1
for all $S \in \mathcal{L}_2$ **do**
  $T_S \leftarrow$ LaminarSpanTree($\mathcal{L}(S), D$)
$T \leftarrow T_1$ with each vertex corresponding to $S \in \mathcal{L}_2$ replaced by T_S, and each edge AB in T_1 replaced by an edge from $E_{G[\mathcal{L}]}(A, B)$
return T

In step 13, an arbitrary edge from $E_{G[\mathcal{L}]}(A, B)$ can be used. Figure 2 illustrates the algorithm.

The next lemma bounds the congestion of the spanning tree constructed in step 13 by combination of T_1 with the trees T_S.

▶ **Lemma 13.** *Let Q be a quotient of G and T a spanning tree of Q. Let $a \in V_Q$ have bag $A = \{a\}^*$. Refine A into a connected partition $\mathcal{P}_A$, and let Q' be the refined quotient of G. Choose a spanning tree T_A of $J_A = G[A]/\mathcal{P}_A$. Replace a in T by T_A, attaching every old incident tree edge to the cell containing the endpoint of its selected original edge. The resulting graph T' is a spanning tree of Q'. For every new tree edge e inside A and every vertex $b \in V_{J_A}$,*

$$c_{Q',T'}(e) \le c_{J_A,T_A}(e) + \ell_{Q,T}(a), \tag{3.2}$$

$$\ell_{Q',T'}(b) \le \ell_{J_A,T_A}(b) + \ell_{Q,T}(a). \tag{3.3}$$

Surviving old tree edges and other old vertices retain their loads. The assertions also hold for simultaneous expansion of disjoint bags, and for any connected positive-weight graph in place of G.

Proof. By construction, T' is a spanning tree. Newly visible graph edges inside A have both endpoints there; their routes stay in T_A and give its local loads. Any previously visible graph edge whose new route uses an edge or vertex inside A had a quotient route visiting a. Their total weight is therefore at most $\ell_{Q,T}(a)$. Projecting routes onto T also proves that other old loads do not change. The same argument applies to simultaneous expansions. ◀

The following lemma is a slight generalization of Lemma 9; its proof is in the Appendix.

▶ **Lemma 14** (Internal trees). *Let F be the local graph at a call of Algorithm 2 with $D \ge 4k$. For every nonempty $C \subseteq V_F$ such that $F[C]$ is connected and $d_F(v) \le D$ for all $v \in C$,* HereditarySpanTree(F) *constructs a spanning tree of $F[C]$ with edge and vertex loads, measured in $F[C]$, at most*

$$O\Big(k\big(D + b_F(C)\big)L^{3/2}\Big). \tag{3.4}$$

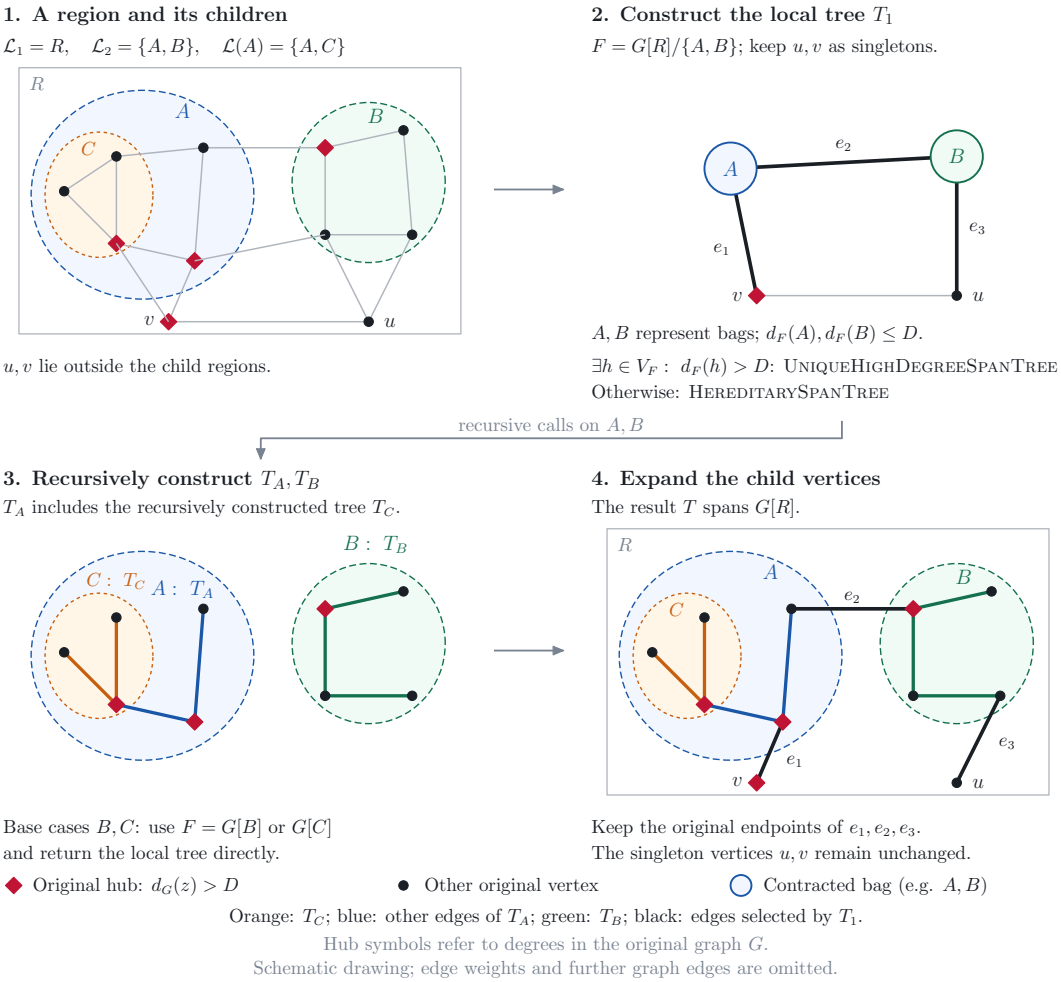


Figure 2 Algorithm 3 on a region R with children A, B and a further child $C \subset A$. Construct the spanning tree T_1 for $F = G[R]/\{A, B\}$, recursively obtain the spanning trees for A and B, and then replace the vertices A and B in T_1 by their spanning trees. The singletons u, v remain unchanged. Red diamonds mark hubs of the original graph G; the large circles in the quotient represent contracted regions.

Thus, if in addition $b_F(C) = O(D)$, then HereditarySpanTree(C) *returns a spanning tree with edge and vertex loads $O(kDL^{3/2})$.*

3.3 Spanning Trees for Subgraphs with one High Degree Vertex

In this subsection, F is a connected positive-integer-weighted graph obtained as a quotient of a subgraph of G, and $U \subsetneq V_F$ is a non-empty vertex set. For a partition $\mathcal{P}$ of U and $C \subseteq U$, define

$$x_{\mathcal{P}}(C) = \omega_F\big(\{uv \in E_F : u, v \in C \text{ belong to different parts of } \mathcal{P}\}\big) \ ;$$

$x_{\mathcal{P}}(C)$ measures how much the set C intersects with the boundaries of the partition $\mathcal{P}$. In this subsection, we first describe how to obtain a "nice" partition of a subset of vertices, and then, how to use the partition to construct a spanning tree of $F[U]$ with bounded congestion.

▶ **Theorem 15** (Partition with controlled boundaries). *Let $N = |V_F|$ be the number of vertices in F, $M = \sum_{e \in E_F} \omega_F(e)$ be the total edge weight of F, β, b_0, d and B_0 parameters satisfying $\beta = \gamma \log^{3/2}(N+M)$, for a sufficiently large constant $\gamma > 0$ and $B_0 = \max\{d, \lceil \beta b_0 \rceil\}$. If every vertex of U has degree at most d, then one can construct, by a randomized polynomial-time algorithm, a partition $\mathcal{P}$ of U into connected sets such that*

$$b_F(A) \le 2B_0 \qquad \forall A \in \mathcal{P}, \tag{3.5}$$

$$x_{\mathcal{P}}(C) \le 4\beta b_F(C) \qquad \forall C \subseteq U,\ b_F(C) \le b_0\ . \tag{3.6}$$

In the proof of Theorem 15, we will need the following algorithm following from the bisection results of Feige and Krauthgamer [4] (Subsections 5.2 and 5.4); for completeness, we describe the reduction in Appendix A.4.

▶ **Proposition 16** (Heavy set with small boundary [4]). *Let Q be a connected undirected graph on N vertices, with nonnegative integer edge capacities and a distinguished vertex o. Let ν be a nonnegative polynomially bounded integer vertex weight with $\nu(o) = 0$; set $W = \nu(V_Q)$, $\beta = O(\log^{3/2}(N+W))$, and let $1 \le \ell \le W$ be an integer. There is a randomized polynomial-time algorithm which finds a set $A \subseteq V_Q \setminus \{o\}$ satisfying*

$$\nu(A) \ge \ell, \qquad b_Q(A) \le \beta \min\{b_Q(C) : o \notin C,\ \nu(C) \ge \ell\}\ .$$

Note that the set A may be disconnected.

Proof of Theorem 15. For a partition $\mathcal{P}$ of U, we define its *potential* as

$$\Phi(\mathcal{P}) = \sum_{A \in \mathcal{P}} b_F(A) = b_F(U) + 2x_{\mathcal{P}}(U) \tag{3.7}$$

We start the construction with $\mathcal{P} = \mathcal{P}_0 = \{\{u\} : u \in U\}$ (i.e., each vertex has its own part in $\mathcal{P}$). Note that at this point, for each $A \in \mathcal{P}$, $b_F(A) \le 2B_0$; this a property, called the *boundary invariant*, that we will maintain. We aim at decreasing the potential as much as possible, and we will do so by a sequence of local changes, called swaps. For $Z \subseteq U$, a *Z-swap of $\mathcal{P}$* is an operation that changes $\mathcal{P}$ into a new partition $\mathcal{P}'$ as follows: For each $A \in \mathcal{P}$ that has a non-empty intersection with Z, A is removed from $\mathcal{P}$, and all connected components of $F[A \setminus Z]$ are added to $\mathcal{P}$, and also all connected components of $F[Z]$ are added to $\mathcal{P}$. Observe that only edges that i) were previously inside a part $A \in \mathcal{P}$, and ii) belong to $\delta_F(Z)$, become edges between parts of $\mathcal{P}'$, while all edges inside Z that cross different parts of $\mathcal{P}$ will not cross any boundary of $\mathcal{P}'$. Hence,

$$\begin{aligned}\Delta\Phi = \Phi(\mathcal{P}') - \Phi(\mathcal{P}) &= 2\big(\omega_F(\text{edges inside any } A \in \mathcal{P} \text{ crossing } Z) - x_{\mathcal{P}}(Z)\big) \\ &\le 2\big(b_F(Z) - x_{\mathcal{P}}(Z)\big).\end{aligned} \tag{3.8}$$

A *plain swap* is a Z-swap performed on a set Z satisfying

$$b_F(Z) \le B_0, \qquad x_{\mathcal{P}}(Z) > b_F(Z). \tag{3.9}$$

After a swap operation, the boundary invariant might be violated: there might be $A \in \mathcal{P}'$ with $b_F(A) > 2B_0$. To make it possible to fix it, we maintain for every $A \in \mathcal{P}'$ a connected set Q_A, called an *enclosing set of A*, with the property

$$A \subseteq Q_A \subseteq U, \qquad b_F(Q_A) \le B_0. \tag{3.10}$$

At the very beginning (i.e., for the trivial partition into singletons) we set, for each $A \in \mathcal{P}$, $Q_A = A$. After performing a Z-swap, we define the enclosing sets for the new sets in the partition as follows: for each connected component of $A \setminus Z$, considered earlier, we define its enclosing set as Q_A (i.e., it inherits the enclosing set of A), and for each connected component C of Z, we set $Q_C = C$.

A *repair swap* is a Q_A-swap for a set A with $b_F(A) > 2B_0$. Since A is a single part of $\mathcal{P}$ and $Q_A \setminus A$ is disjoint from A, the edges between A and $Q_A \setminus A$ join different parts in the partition before performing this swap,

$$x_{\mathcal{P}}(Q_A) \geq \omega_F(A, Q_A \setminus A) \geq b_F(A) - b_F(Q_A). \tag{3.11}$$

Thus, performing the Q_A-swap changes the potential (due to 3.8 and 3.11) by

$$\Delta\Phi \leq 2(b_F(Q_A) - x_{\mathcal{P}}(Q_A)) \leq 2[b_F(Q_A) - (b_F(A) - b_F(Q_A))] = 4b_F(Q_A) - 2b_F(A) < 0.$$

Both plain and repair swaps decrease the nonnegative integer potential by at least two. Initially, it was at most $2M$, so there will be at most M operations.

To complete the description of the sequence of swaps that we use, a few more terms are needed. For each $v \in U$, let $A(v)$ be its part in the current partition, and define

$$\nu(v) = \sum_{\substack{vw \in E_F \\ w \in U,\ A(w) \neq A(v)}} \omega_F(vw);$$

$\nu(v)$ is the weight of edges from v to other parts of the current partition of U. For $C \subseteq U$, put $\nu(C) = \sum_{v \in C} \nu(v)$. Then

$$\nu(C) = 2x_{\mathcal{P}}(C) + \nu_\partial(C), \tag{3.12}$$

where $\nu_\partial(C)$ is the total weight of edges between C and $U \setminus C$ that connect different parts of the current partition; note that for each $C \subseteq U$, $\nu_\partial(C) \leq b_F(C)$.

Let $F_o = F/\{V_F \setminus U\}$, and let o be the vertex obtained by the contraction of $V_F \setminus U$. Note that $\forall A \subseteq U$, $b_{F_o}(A) = b_F(A)$.

Here is the complete algorithm constructing the desired partition. Start with $\mathcal{P} = \mathcal{P}_0$, and then repeat:

1. While there is $A \in \mathcal{P}$ with $b_F(A) > 2B_0$, perform Q_A-swap.
2. Compute the weight function ν. If $\nu(U) = 0$, STOP.
3. Set $\nu(o) = 0$. For every q equal to a power of two, $q \leq \nu(U)$, apply Proposition 16 on F_o with the weight function ν and $\ell = q$.
4. Swap in any returned set satisfying (3.9) and repeat. If none satisfies it, STOP.

The repair-swaps ensure the property (3.5). For a contradiction assume that the property (3.6) is not satisfied when the algorithm stops, that is, some $C \subseteq U$ satisfies $b_F(C) \leq b_0$ and $x_{\mathcal{P}}(C) > 4\beta b_F(C)$. The strict positivity of $x_{\mathcal{P}}(C)$ implies, by (3.12), that $\nu(C) > 0$. There is a value q that is a power of two, was tested by the algorithm, and satisfies $\nu(C)/2 < q \leq \nu(C)$. Thus, the algorithm of Proposition 16 finds a set Z satisfying $b_F(Z) \leq \beta b_F(C) \leq B_0$ and

$$\nu(Z) \geq q > \frac{\nu(C)}{2} \overset{(3.12)}{\geq} x_{\mathcal{P}}(C) > 4\beta b_F(C) \geq 4b_F(Z).$$

Combining $\nu(Z) > 4b_F(Z)$ with $\nu_\partial(C) \leq b_F(C)$ and equation (3.12) gives $x_{\mathcal{P}}(Z) > \frac{3}{2} b_F(Z)$, contradicting that no set was found in step 4 (note that if $\nu(U) = 0$, no edges join different parts inside U, and the conclusion is immediate).

There are at most M swaps, one final unsuccessful run of the algorithm from Proposition 16, and $O(\log(2M))$ calls of Proposition 16 per search. All weights and the parameter ℓ are of size at most $2M$. Amplify each call by $O(\log(N+M))$ independent repetitions and take a union bound over these polynomially many calls, conditional on the preceding choices. With high probability every call has the claimed approximation factor, so the stopping argument applies simultaneously to all sets C. Since $d \geq 1$, the parameters give $B_0 = O(d + b_0 \log^{3/2}(N+M))$ and $4\beta = O(\log^{3/2}(N+M))$. ◀

▶ **Lemma 17** (Constructing a spanning tree from Theorem 15). *Let F be a quotient of G constructed by Algorithm 2, with $b_G(V_F^*) \leq D$ for $D \geq 4k$, and $h \in V_F$ be an original singleton such that $d_F(v) \leq D$ for all $v \in V_F \setminus \{h\}$. There is a randomized polynomial time algorithm, which we call later* UniqueHighDegreeSpanTree(F, h, D)*, that constructs a spanning tree T_1 of F satisfying*

$$c_{F,T_1}(e) = O(kDL^4) \qquad \text{for every } e \in E(T_1), \tag{3.13}$$

$$\ell_{F,T_1}(v) = O(kDL^4) \qquad \text{for every } v \in V_F \setminus \{h\}. \tag{3.14}$$

Proof. If $V_F = \{h\}$, return the empty tree. Otherwise apply Theorem 15 in F to $U = V_F \setminus \{h\}$, with $d = D$ and $b_0 = D$ to get the partition $\mathcal{P}$. Since F has at most n vertices and total edge weight at most $m < n^2$, the partition $\mathcal{P}$ satisfies

$$b_F(A) = O(DL^{3/2}) \qquad \forall A \in \mathcal{P}, \tag{3.15}$$

$$x_{\mathcal{P}}(C) \leq O(L^{3/2}) b_F(C) \qquad \forall C \subseteq U,\ b_F(C) \leq D\ . \tag{3.16}$$

For $v \in U$, let $A(v)$ be the part $A \in \mathcal{P}$ such that $v \in A$, and let

$$b(v) = \sum_{\substack{vw \in E_F \\ w \notin A(v)}} \omega_F(vw)$$

be the sum of weights of edges to neighbors of v *not* in $A(v)$. Let $F_{\mathcal{P}} = (F-h)/\mathcal{P}$ be the quotient of $F-h$ defined by $\mathcal{P}$, and define the *supply* of every vertex $A \in \mathcal{P}$ as $b(A) = \sum_{v \in A} b(v)$; we say that $b(v)$ is the *contribution of* v to the supply $b(A)$ of A. Consider the following flow instance $I_{F,\mathcal{P}}$ on $F_{\mathcal{P}}$: Every vertex $A \in \mathcal{P}$ is a source with supply $b(A)$, and the neighbors of h in $F_{\mathcal{P}}$ are the sinks. Every edge has an infinite capacity, and every vertex has capacity c for a value $c > 0$ specified later; this capacitated graph is denoted $F_{\mathcal{P}}^c$. The task is to find in $F_{\mathcal{P}}^c$, if exists, a flow that delivers all the supply to the sinks, subject to the capacity constraints.

▷ Claim 18. There exists $c = O(kDL^3)$ such that the instance $I_{F,\mathcal{P}}$ is feasible in $F_{\mathcal{P}}^c$.

On the high level, the proof of the claim goes along similar lines as the proof of the min-max connected cut Theorem 23. The significant difference is that now we exploit the existence of a spanning tree of G with congestion k; together with the partition of Theorem 15, this makes it possible to get the flow.

Proof. For the analysis, let $R \in \mathcal{L}$ be the original (vertex) region of the call, that is, $F = G[R]/\mathcal{L}(R)$ and $b_G(R) \leq D$. Recall the convention that contracting an isolating set S_z retains the name z for the resulting vertex, and for every component P of $T^* \setminus \{h\}$, define $C_P = P \cap U$. We call P a *branch* (of T^*). Since h is an original singleton, the sets C_P partition the vertex set U, and by construction, for every branch P,

$$b_F(C_P) \leq b_G(P) \leq k\ . \tag{3.17}$$

Attribute the contribution $b(v)$ of the vertex v to the branch P such that $v \in P$. The total contributions attributed to a branch P is, exploiting (3.16)–(3.17),

$$\sum_{v \in C_P} b(v) \leq 2x_{\mathcal{P}}(C_P) + b_F(C_P) = O(kL^{3/2}) \ . \tag{3.18}$$

Call a branch P *R-external* if for some $v \in C_P$, the v–h path in T^* leaves R; all other branches are *R-internal*. There are at most $b_G(R) \leq D$ R-external branches: each contains a distinct tree edge leaving R. Define the *projection of P on $\mathcal{P}$* as the set $P_{\mathcal{P}} = \{A \in \mathcal{P} : A^* \cap P \neq \emptyset\}$.

For every R-internal branch P, for each $v \in C_P$, project the unique v–h path in T^* to $F_{\mathcal{P}}$, erase loops, and use the remaining path to route $b(v)$ units (which is the contribution of v to P) until the first vertex of $N_{F_{\mathcal{P}}}(h)$. By the properties of $\mathcal{P}$ and R, each part $A \in \mathcal{P}$ intersects with projections of at most $b_G(A^*) \leq b_F(A) + b_G(R) = O(DL^{3/2})$ branches.

For every R-external branch P, route the contribution of all vertices in C_P along arbitrary simple paths to h in the connected graph $F_{\mathcal{P}}$, stopping at the first neighbor of h.

We conclude that the total flow going in any $A \in \mathcal{P}$ originates from at most $O(DL^{3/2}) + D$ branches, therefore, by equation (3.18), its size is at most $O(kL^{3/2})(O(DL^{3/2}) + D) = O(kDL^3)$, and this is also a bound on the load of any vertex other than h. ◀

The flow of Claim 18 can be obtained by repeated max-flow computations: Let l be a guess on the maximum load from the claim (obtained, e.g., by trying all integer values for it, smaller than $\sum_{A \in \mathcal{P}} b_F(A) \leq 2m$, starting with 1, or by binary search on $[1, 2m]$). For each such guess, define for every node, other than h, its capacity as l and try to find a feasible solution (i.e., respecting the node capacities) to the flow instance $I_{F,\mathcal{P}}$. The flow corresponding to the smallest value of l for which the instance is feasible, is the desired flow. Note that for the capacity $l = 2m$, the instance is feasible; thus, the search always terminates.

Apply now the confluent-flow algorithm of Theorem 5 on the flow from Claim 18; the loss factor is $1 + \ln |N_{F_{\mathcal{P}}}(h)| = O(L)$. This gives us a confluent flow f satisfying the instance $I_{F,\mathcal{P}}$ with maximum node load $O(kDL^4)$; clearly, the same bound applies also to the maximum edge load. Recall that the flow f is realized by a collection of node disjoint trees. We combine them into a spanning tree τ of $F_{\mathcal{P}}$ by adding all edges between the sinks $N_{F_{\mathcal{P}}}(h)$ of $I_{F,\mathcal{P}}$ and the vertex h. The construction of τ derived from the flow for the instance $I_{F,\mathcal{P}}$ ensures that the congestion of τ as well as the load of every vertex other than h is $O(kDL^4)$.

Every vertex in a part A of $\mathcal{P}$ has degree at most D in F. Thus, for each $A \in \mathcal{P}$, we can construct a spanning tree for $F[A]$ by Lemma 14; the bound on the edge and vertex loads of these local spanning trees is

$$O\Big(k(D + b_F(A))L^{3/2}\Big) = O(kDL^3).$$

Now substitute these "local" trees for the vertices of τ, realizing tree edges of $F_{\mathcal{P}}$ by edges of F as follows: For each edge AB of τ, use any edge from $E_F(A, B)$. The result is a spanning tree of F. By Lemma 13 and the above bounds, the maximum edge load and vertex load for all vertices but h, is at most $O(kDL^3) + O(kDL^4) = O(kDL^4)$. ◀

At every call of Algorithm 2, Lemma 14 applies if all degrees in F are at most D. Otherwise the unique vertex of degree greater than D is an original singleton, and Lemma 17 applies to the call (F, h, D). Thus the chosen local tree has edge loads and vertex loads at all non-singleton cells bounded by $O(kDL^4)$. This includes the root and the leaves of the laminar family.

3.4 Approximation Guarantee

▶ **Lemma 19** (Adding the costs of the levels). *For every $D \geq 4k$, the preceding construction returns, with high probability, a spanning tree T of G with*

$$c(G,T) = O(kDL^5).$$

Proof. At every stage, the current bags partition V_G and the current tree is a spanning tree of their weighted quotient. This follows inductively from Lemma 13. The local graph exposed when A is expanded is exactly $G[A]/\mathcal{L}(A)$, independently of how vertices outside A are currently grouped. The local trees were constructed using only these induced graphs. The comparison with original cuts was needed for their analysis; the expansion accounting uses the current quotient.

Track edge loads and vertex loads only at non-singleton bags. Each local tree bounds these two quantities by $O(kDL^4)$, at every call, including the root: use Lemma 14 when all local degrees are at most D, and Lemma 17 otherwise. The high-degree vertex of Lemma 17 is an original singleton, so it is never expanded and its vertex load need not be bounded.

Suppose the two tracked quantities are bounded by $O(jkDL^4)$ after j processed levels. For an expanded bag A, the local load and its previous vertex load add in (3.2)–(3.3), giving $O((j+1)kDL^4)$ for new edges and new non-singleton cells. Surviving old loads do not increase. More explicitly, fix a common absolute constant for the local bounds; the same constant multiplied by j works in this induction. The family has $O(L)$ levels, so the final edge loads are $O(kDL^5)$. ◀

For the final approximation, compare the least-congested tree obtained from Algorithm 2 over all tested thresholds with the tree of congestion at most $8\sqrt{mn}$ from Lemma 2. Return the better tree.

▶ **Theorem 20.** *There is a randomized polynomial-time algorithm with approximation ratio*

$$O\left(\min\left\{kL^5, \frac{\sqrt{mn}}{k}\right\}\right) \leq O(n^{2/3}L^{5/3}) = \widetilde{O}(n^{2/3}). \tag{3.19}$$

Proof. Some tested threshold satisfies $4k \leq D < 8k$, because $1 \leq k \leq m$. Lemma 12 guarantees progress for this threshold. All other guesses either return a family after at most $n-1$ contractions or propagate FAIL; the local routines also terminate in polynomial time for these guesses. Lemma 19 gives a candidate of congestion $O(k^2L^5)$. Comparing it with the $8\sqrt{mn}$ candidate proves the first bound in (3.19). As in Section 1, $m \leq k(n-1) \leq kn$. Consequently the ratio is at most

$$O\Big(\min\{kL^5, n/\sqrt{k}\}\Big) \leq O\Big((kL^5)^{1/3}(n/\sqrt{k})^{2/3}\Big) = O(n^{2/3}L^{5/3}).$$

Each successful contraction contains at least two current vertices, so at most $n-1$ contractions occur for any guess. Every local graph has at most n vertices and total weight at most m. The partition algorithm, node-capacitated flow, confluent rounding, and the hereditary-bisection routine are therefore polynomial-time operations also for guesses below $4k$; their congestion guarantees are needed only for a successful threshold.

There are polynomially many randomized calls over all guesses. Amplify the bisection and quota calls using $O(\log n)$ independent repetitions, retaining the least-cost feasible output. A conditional union bound over the calls makes all required guarantees hold with probability at least $1 - n^{-c}$, for any fixed $c > 0$. The stopping argument in Theorem 15 then guarantees its inequalities simultaneously for all subsets, without testing them. Evaluating the congestion of each candidate ensures that other guesses cannot worsen the returned solution. ◀

4 Hardness within $2-\varepsilon$

Next we prove the following hardness result.

▶ **Theorem 21.** *For every fixed $\varepsilon \in (0,1)$, there is no polynomial-time $(2-\varepsilon)$-approximation algorithm for spanning tree congestion, unless* P = NP.

The proof is by reduction from a restricted version of the vertex disjoint path problem: given a graph Q and p pairs $(s_1,t_1),\dots,(s_p,t_p)$ of vertices, can all p pairs be joined by mutually vertex-disjoint paths? The problem is NP-complete even when the graph has maximum degree three, all $2p$ terminals are distinct, and a vertex has degree one if and only if it is a terminal [10, proof of Theorem 1].

We may also require Q to be connected. First reject instances having a pair in different components, and discard components without terminals. In each remaining component subdivide an edge twice and join consecutive components by an edge between distinct subdivision vertices, forming a chain of components. Degrees remain at most three and terminals retain degree one. A simple path between terminals in one old component cannot leave that component: it would have to cross a joining edge twice. The transformation therefore preserves the answer. Instances with no terminal pair can be decided directly. Write $N_Q = |V(Q)|$, $M_Q = |E(Q)|$, and $B = \max\{M_Q, 2\}$.

Two observations explain the source restriction. Edge-disjoint paths for distinct degree-one terminals in a graph of maximum degree three are vertex-disjoint: a shared internal vertex would require four incident edges, and a terminal cannot be internal to another path. Conversely, the union of vertex-disjoint paths is a forest and can be extended to a spanning tree of a connected graph.

For a positive-integer-weighted graph H, let $c_w(H,R,e)$ be the total weight crossing the fundamental cut of e in a spanning tree R. Write $c_w(H,R) = \max_{e\in E(R)} c_w(H,R,e)$ and $\mathrm{STC}_w(H) = \min_R c_w(H,R)$.

Fix an integer $A \geq 2$. Give the edges of Q weight one. For every pair (s_i,t_i), add A distinct vertices $w_{i1},\dots,w_{iA}$ and edges

$$s_i w_{ij} \text{ of weight } 1, \qquad w_{ij} t_i \text{ of weight } A+1 \qquad (1 \leq j \leq A). \tag{4.1}$$

Call the resulting weighted graph H_A. The two-edge path $s_i w_{ij} t_i$ is *full* in a spanning tree if both its edges belong to that tree. At most one such path for a fixed pair can be full, since two would form a cycle.

▶ **Lemma 22.** *If the disjoint-path instance is a YES instance, then $\mathrm{STC}_w(H_A) \leq A+B$. From any spanning tree of H_A with congestion less than $2A$, one can extract a vertex-disjoint routing of all pairs in Q in polynomial time. Consequently, a NO instance satisfies $\mathrm{STC}_w(H_A) \geq 2A$.*

Proof. For a YES instance, extend its vertex-disjoint paths to a spanning tree T_Q of Q, and attach every w_{ij} to t_i by its heavy edge. This gives a spanning tree R of H_A. Consider $e \in E(T_Q)$ and the corresponding cut $(S, V(Q)\setminus S)$. All auxiliary vertices lie on the side of their t_i. Thus a gadget contributes A if its terminals are separated and zero otherwise. At most one prescribed path uses e, and so

$$c_w(H_A,R,e) = |\delta_Q(S)| + A\,|\{i : e \in E(P_{T_Q}(s_i,t_i))\}| \leq M_Q + A.$$

Every other tree edge isolates one auxiliary vertex and has load $1+(A+1) = A+2$. Hence $c_w(H_A,R) \leq A+B$.

Now let R be any spanning tree of H_A with congestion below $2A$. Suppose $s_i w_{ij} t_i$ is full. Deleting its heavy edge separates s_i from t_i. The heavy edge contributes $A+1$ to this cut, and each of the other $A-1$ two-edge gadget paths contributes at least one. The heavy tree edge therefore has load at least $2A$, a contradiction. It follows that every auxiliary vertex is a leaf of R, and removing them leaves a spanning tree T_Q of Q.

If two prescribed paths in T_Q used the same edge e, deleting e in R would separate both pairs. Each of the A gadget paths for each pair must cross this cut and contributes at least one. Together with e itself, the load would be at least $2A+1$. Consequently these paths are edge-disjoint, and the source restriction makes them vertex-disjoint. Removing auxiliary leaves and finding the paths takes polynomial time. ◀

Lemma 22 gives

$$\begin{aligned} &\text{YES:} \quad \mathrm{STC}_w(H_A) \le A + B, \\ &\text{NO:} \quad \mathrm{STC}_w(H_A) \ge 2A. \end{aligned} \tag{4.2}$$

Proof of Theorem 21. Fix $\varepsilon \in (0,1)$ and take $A = \lceil 2B/\varepsilon \rceil + 1$. Then

$$2A - (2-\varepsilon)(A+B) = \varepsilon A - (2-\varepsilon)B > 0.$$

A $(2-\varepsilon)$-approximation would return a tree of congestion below $2A$ on every YES instance, whereas no such tree exists on a NO instance. Evaluating the returned tree's congestion would decide the NP-complete source problem. To eliminate integer weights, we use the standard reduction that replaces an edge with weight $w > 1$ by w internally disjoint paths [9]. The construction is polynomial. ◀

AI Disclosure.

The idea of using connected min-max multiway partition to approximate spanning tree congestion is from the authors, specifically establishing a reduction when the graph has only one vertex with unbounded degree, with approximation ratio dependent on the second largest degree D. The main results in this paper were subsequently discovered by GPT-6 Astra; this includes the $\tilde{O}(n^{8/9})$ approximation algorithm, the log-factor improvement in Lemma 2 combining [2, 5], and Theorems 1, 4, 21.

A Omitted Proofs

A.1 Lemma2

Proof of Proposition 2. The construction in [2, Theorem 10 and Algorithm 1] already gives this congestion. Its non-polynomial step is a connected partition with bounded degree sums. We show how to compute that partition. Let H be an ℓ-vertex-connected simple graph with s vertices, M edges, and distinct prescribed terminals $t_1, \ldots, t_\ell$. Set

$$A = \left\lceil \frac{2M}{\ell} \right\rceil, \qquad w(v) = d_H(v) \quad (v \notin \{t_1, \ldots, t_\ell\}), \qquad c_i = A - d_H(t_i).$$

Since $2M/\ell \ge s > \Delta(H)$, the capacities c_i are nonnegative and their sum covers all nonterminal weights. Replace each edge by opposite arcs and delete outgoing terminal arcs. Menger's theorem gives ℓ paths from every nonterminal to distinct terminals, disjoint except

at their starting vertex. Thus [5, Theorem 3] applies and computes connected parts V_i, each containing t_i and no other terminal, with

$$\sum_{v \in V_i} d_H(v) \le A + \max_{v \notin \{t_1,\dots,t_\ell\}} w(v) - 1 \le \left\lceil \frac{2M}{\ell} \right\rceil + \Delta(H) - 1 \le \frac{4M}{\ell}.$$

This is the degree-sum guarantee needed in line 15 of Algorithm 1 in [2]. Replacing that step preserves their congestion analysis and makes the whole construction polynomial-time. ◀

A.2 Lemma 3

Proof of Lemma 3. The edge-congestion bound is from the paper [6, Algorithm 1, Lemma 3.2, and Theorem 1.1]. Its hereditary bisection parameter is at most B: deleting edges from an induced subgraph cannot increase its minimum balanced-cut cost. Integer weights are represented by parallel unit edges.

For the vertex bound, just note that each recursive cut has weight $O(B\sqrt{\log n})$, and its sides have at most three quarters of the current vertices. The recursive children are the connected components of the two sides. Every demand not internal to one child crosses this cut, so its total contribution to any vertex, including endpoints, is at most the cut weight. A fixed vertex lies on a chain of $O(\log n)$ recursive subproblems, ending at a singleton with no internal demand. Summing gives $O(B \log^{3/2} n)$. ◀

A.3 Lemma 14

Proof of Lemma 14. For $X \subseteq C$, $|X| \ge 2$, mark the vertices of X in an optimum spanning tree T^* of G and choose a centroid x: each component of $T^* - x$ has at most $|X|/2$ marks. If $x \in C^*$, select $v \in C$ such that $x \in \{v\}^*$. Apart from the possible component containing v, each marked component has a distinct first-exit tree edge from $\{v\}^*$ toward a marked vertex outside this bag. There are at most $b_G(\{v\}^*) \le 2D$ such edges. If $x \notin C^*$, each marked component has a distinct first-entrance tree edge into $C^{\uparrow}$, giving at most $b_G(C^*) \le b_F(C) + D$ marked components.

Choose whole components carrying between $|X|/3$ and $2|X|/3$ marks. A component with at least $|X|/3$ marks suffices; otherwise accumulate components until this threshold is reached. There are enough marks outside x, since $|X| - 1 \ge |X|/3$. Each component has original boundary at most k, so their union has boundary $O(k(D + b_F(C)))$. Intersecting this union with X gives a balanced cut of $F[X]$ with no greater cost. Deleting edges only decreases this cost, so the same bound holds for the hereditary bisection of $F[C]$. Lemma 3 completes the proof. ◀

A.4 Proposition 16

Proof of Proposition 16. Consider the graph Q, $o \in V_Q$, W, the ν-weight, ℓ and as described in Proposition 16. We modify Q into a larger w-weighted graph on which we apply the bisection approximation of Feige and Krauthgamer. Start with the graph Q and add to it $W - \ell$ isolated vertices of w-weight one and W isolated vertices of w-weight zero. Keep every edge of Q with its capacity, give the vertex o weight $w(o) = \ell$, and to every other vertex $v \in V_Q, v \ne o$, $w(v) = \nu(v)$. The total vertex weight is now $2W$. A weight-W bisection side excluding o contains vertices from Q of w-weight at least $W - (W - \ell) = \ell$; the ν-weight of the set is the same. Conversely, an admissible original set of weight $s \ge \ell$ can be extended

with $W - s$ unit w-weight isolated vertices to obtain such a side. Cut costs and optimum values in both graphs are the same.

Apply the edge-cost and integer-vertex-weight versions of the algorithm by Feige and Krauthgamer (Sections 5.1–5.2). The added zero-weight vertices ensure that the vertex weights are bounded by the extended order. This yields the claimed $O(\log^{3/2}(N + W))$ factor. ◀

B Approximation of the connected min–max multiway cut

This appendix proves Theorem 4. We first show how to make a terminal partition connected. The estimate keeps track of the total boundary, which gives the edge-sensitive bound used in the simpler STC algorithm.

▶ **Theorem 23** (Connecting a terminal partition). *Let F be a connected graph with positive integer edge weights, and let $\mathcal{A}$ be a partition into $t \geq 2$ parts, each containing one prescribed terminal. Put*

$$B = \max_{A \in \mathcal{A}} b_F(A), \qquad S = \sum_{A \in \mathcal{A}} b_F(A).$$

In polynomial time one can find a connected one-terminal partition with maximum boundary $O(\sqrt{SB}\log t)$.

Proof. Split each initial part into its connected components and contract these components, obtaining a weighted graph J. Each vertex x of J has degree $d_J(x) \leq B$. The components of one initial part have no edges between them, so

$$\sum_{x \in V(J)} d_J(x) = S \leq tB.$$

Exactly t of the contracted vertices contain a terminal. Declare them sinks. Give every vertex x the fixed supply $d_J(x)$, and put $\eta = \lceil \sqrt{S/B} \rceil$. Both S and B are positive.

Give each edge capacity η times its weight and try to route all supplies to the sinks. This is a single-commodity maximum-flow computation: a new source injects each supply, both orientations of an edge have its capacity, and each sink has an arc of capacity S to a new target. If all supply cannot be routed, a minimum cut gives a sink-free set U_j in the current graph J_j such that

$$\eta b_{J_j}(U_j) < \sum_{x \in U_j} d_J(x). \tag{B.1}$$

Indeed, the source cut has capacity at most the total remaining supply, whereas cutting any sink-to-target arc costs S; a strictly violating set therefore contains no sink. Delete U_j and repeat, always retaining the original supplies $d_J(x)$. At most $|V(J)|$ iterations leave a feasible subgraph on a set W. Every sink remains, so W is nonempty.

Let U be the union of the deleted sets. They are disjoint, and any edge from U to W was counted when its endpoint in U was deleted. Summing (B.1) therefore gives

$$\omega_J(U, W) \leq S/\eta, \tag{B.2}$$

where $\omega_J(U, W)$ is the total edge weight between U and W.

Stop the feasible flow at the first sink it reaches and cancel cycles. Incoming flow at x is at most $\eta d_J(x)$, since the sum of the capacities of the incident edges is at most that quantity.

Thus its node load, including its own supply, is at most $(\eta+1)B$. By the confluent-flow theorem in [3, Theorem 20], this flow can be converted in polynomial time into a flow in which all flow leaving a non-sink uses one outgoing edge, with node loads at most $(1+\ln t)(\eta+1)B$. The bound includes the loads at sinks and the supplies at all vertices. The positive supplies ensure that all vertices of W belong to the resulting connected groups, each containing exactly one sink. The total supply in a group is its sink load, and it bounds that group's boundary in the original quotient J.

Attach each whole connected component of $J[U]$ to one adjacent group. Such a neighbor in W exists because J is connected. There are no edges between different components of $J[U]$. Therefore, for any final group, the increase in its boundary is at most the entire interface $\omega_J(U,W)$, rather than the sum of the original degrees of the attached vertices. By (B.2), the resulting maximum boundary is at most

$$(1+\ln t)(\eta+1)B + S/\eta = O(\sqrt{SB}\log t).$$

No deleted vertex contains a terminal. The final groups are connected and have one terminal each; expanding their constituent components gives the required partition of F. ◀

Proof of Theorem 4. Denote $B^* = OPT$. Use the min–max multiway-cut algorithm from [1, Theorem 1.4]. It produces a one-terminal partition, not necessarily connected, with maximum boundary

$$B \le O\Big(\sqrt{\log(n)\log(t)}\Big)\, B^* \le O(\log n)B^*.$$

An expected approximation guarantee suffices here: Markov's inequality gives constant success probability, and $O(\log n)$ independent repetitions, selecting the smallest actual maximum boundary, give this bound with high probability. The optimum without connectivity is at most B^*, so it is a valid comparison value. For the returned partition,

$$B \le S \le tB, \qquad S \le 2M.$$

Apply Theorem 23 for $F = G$. Its maximum boundary is at most

$$O\Big(\min\{\sqrt{t}\,B, \sqrt{MB}\}\log(t)\Big) = \widetilde{O}\Big(\min\{\sqrt{t}\,B^*, \sqrt{MB^*}\}\Big),$$

as claimed. More explicitly, the two bounds are respectively

$$O(\sqrt{t}\,B^*\log^2 n) \quad \text{and} \quad O(\sqrt{MB^*}\log^{3/2} n).$$

For $t = 1$ the whole connected graph is the optimal part, with zero boundary, and no approximation routine is needed. ◀

References

1 Nikhil Bansal, Uriel Feige, Robert Krauthgamer, Konstantin Makarychev, Viswanath Nagarajan, Joseph Naor, and Roy Schwartz. Min-max graph partitioning and small set expansion. *SIAM J. Comput.*, 43(2):872–904, 2014. doi:10.1137/120873996.

2 L. Sunil Chandran, Yun Kuen Cheung, and Davis Issac. Spanning tree congestion and computation of generalized Győri–Lovász partition. In *Proc. of 45th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 107 of *LIPIcs*, pages 32:1–32:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICALP.2018.32.

3 Jiangzhuo Chen, Robert D. Kleinberg, László Lovász, Rajmohan Rajaraman, Ravi Sundaram, and Adrian Vetta. (Almost) tight bounds and existence theorems for single-commodity confluent flows. *Journal of the ACM*, 54(4), 2007. Article 16. doi:10.1145/1255443.1255444.

4 Uriel Feige and Robert Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM Review*, 48(1):99–130, 2006. doi:10.1137/050640904.

5 Mohammad T. Hajiaghayi, Mahdi JafariRaviz, Alireza Kaviani, and Soheil Mohammadkhani. Breaking the exponential barrier: The first polynomial-time algorithm for the Győri–Lovász theorem, 2026. arXiv:2608.30945v1. doi:10.48550/arXiv.2608.30945.

6 Petr Kolman. Approximation of spanning tree congestion using hereditary bisection. *Discrete Mathematics & Theoretical Computer Science*, 28(2), 2026. Article 35. doi:10.46298/dmtcs.16997.

7 Petr Kolman and Hans Raj Tiwary. Min-max connected multiway cut, 2026. arXiv:2602.13861.

8 Jason Li and Debmalya Panigrahi. Approximate Gomory–Hu tree is faster than $n-1$ max-flows. In *Proc. of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, page 1738–1748. Association for Computing Machinery, 2021. doi:10.1145/3406325.3451112.

9 Huong Luu and Marek Chrobak. Better hardness results for the minimum spanning tree congestion problem. *Algorithmica*, 87:148–165, 2025. Preliminary version in *Proc. of 17th International Conference and Workshops on Algorithms and Computation (WALCOM)*, 2023. doi:10.1007/s00453-024-01278-5.

10 Leo van Iersel, Steven Kelk, Georgios Stamoulis, Leen Stougie, and Olivier Boes. On unrooted and root-uncertain variants of several well-known phylogenetic network problems. *Algorithmica*, 80(11):2993–3022, 2018. doi:10.1007/s00453-017-0366-5.