Spanning Tree Congestion and Computation of Generalized Győri-Lovász Partition

L. Sunil Chandran[1]

Department of Computer Science and Automation, Indian Institute of Science, India
sunil@csa.iisc.ernet.in

Yun Kuen Cheung[2]

Max Planck Institute for Informatics, Saarland Informatics Campus, Germany
ycheung@mpi-inf.mpg.de
https://orcid.org/0000-0002-9280-0149

Davis Issac

Max Planck Institute for Informatics, Saarland Informatics Campus, Germany
dissac@mpi-inf.mpg.de
https://orcid.org/0000-0001-5559-7471

Abstract

We study a natural problem in graph sparsification, the Spanning Tree Congestion (STC) problem. Informally, it seeks a spanning tree with no tree-edge *routing* too many of the original edges.

For any general connected graph with n vertices and m edges, we show that its STC is at most $\mathcal{O}(\sqrt{mn})$, which is asymptotically optimal since we also demonstrate graphs with STC at least $\Omega(\sqrt{mn})$. We present a polynomial-time algorithm which computes a spanning tree with congestion $\mathcal{O}(\sqrt{mn} \cdot \log n)$. We also present another algorithm for computing a spanning tree with congestion $\mathcal{O}(\sqrt{mn})$; this algorithm runs in sub-exponential time when $m = \omega(n \log^2 n)$.

For achieving the above results, an important intermediate theorem is *generalized Győri-Lovász theorem*. Chen et al. [8] gave a non-constructive proof. We give the first elementary and constructive proof with a local search algorithm of running time $\mathcal{O}^*(4^n)$. We discuss some consequences of the theorem concerning graph partitioning, which might be of independent interest.

We also show that for any graph which satisfies certain *expanding properties*, its STC is at most $\mathcal{O}(n)$, and a corresponding spanning tree can be computed in polynomial time. We then use this to show that a random graph has STC $\Theta(n)$ with high probability.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners

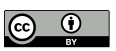
Keywords and phrases Spanning Tree Congestion, Graph Sparsification, Graph Partitioning, Min-Max Graph Partitioning, k-Vertex-Connected Graphs, Győri-Lovász Theorem

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.32

Related Version A full version of this paper is available at `https://arxiv.org/abs/1802.07632`.

[1] This work was done while this author was visiting Max Planck Institute for Informatics, Saarbrücken, Germany, supported by Alexander von Humboldt Fellowship.

[2] Part of the work done while this author was a visitor at the Courant Institute, NYU. The visit was funded in part by New York University.



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 32; pp. 32:1–32:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Graph Sparsification/Compression generally describes a transformation of a *large* input graph into a *smaller/sparser* graph that preserves certain feature (e.g., distance, cut, congestion, flow) either exactly or approximately. The algorithmic value is clear, since the smaller graph might be used as a preprocessed input to an algorithm, so as to reduce subsequent running time and memory requirement. In this paper, we study a natural problem in graph sparsification, the Spanning Tree Congestion (STC) problem. Informally, the STC problem seeks a spanning tree with no tree-edge *routing* too many of the original edges. The problem is well-motivated by network design applications, where designers aim to build sparse networks that meet traffic demands, while ensuring no connection (edge) is too congested. Indeed, the root of this problem dates back to at least 30 years ago under the name of "load factor" [5, 25], with natural motivations from parallel computing and circuit design applications. The STC problem was formally defined by Ostrovskii [21] in 2004, and since then a number of results have been presented.

Two canonical goals for graph sparsification problems are to understand the trade-off between the sparsity of the output graph(s) and how well the feature is preserved, and to devise (efficient) algorithms for computing the sparser graph(s). These are also our goals for the STC problem. We focus on two scenarios: (A) general connected graphs with n vertices and m edges, and (B) graphs which exhibit certain *expanding properties*.

For (A), we show that the spanning tree congestion (STC) is at most $\mathcal{O}(\sqrt{mn})$, which is a factor of $\Omega(\sqrt{m/n})$ better than the trivial bound of m. We present a polynomial-time algorithm which computes a spanning tree with congestion $\mathcal{O}(\sqrt{mn} \cdot \log n)$. We also present another algorithm for computing a spanning tree with congestion $\mathcal{O}(\sqrt{mn})$; this algorithm runs in sub-exponential time when $m = \omega(n \log^2 n)$. For almost all ranges of average degree $2m/n$, we also demonstrate graphs with STC at least $\Omega(\sqrt{mn})$. For (B), we show that the expanding properties permit us to devise polynomial-time algorithm which computes a spanning tree with congestion $\mathcal{O}(n)$. Using this result, together with a separate lower-bound argument, we show that a random graph has $\Theta(n)$ STC with high probability.

For achieving the results for (A), an important intermediate theorem is *generalized Győri-Lovász theorem*, which was first proved by Chen et al. [8]. Their proof uses advanced techniques in topology and homology theory, and is non-constructive. For brevity, we will say "k-connected" for "k-vertex-connected" henceforth.

▶ **Definition 1.** In a graph $G = (V, E)$, a k-connected-partition is a k-partition of V into $\cup_{j=1}^{k} V_j$, such that for each $j \in [k]$, $G[V_j]$ is connected.

▶ **Theorem 2** ([8, Theorems 25, 26])**.** *Let $G = (V, E)$ be a k-connected graph. Let w be a weight function $w : V \to \mathbb{R}^+$. For any $U \subset V$, let $w(U) := \sum_{v \in U} w(v)$. Given any k distinct terminal vertices $t_1, \cdots, t_k$, and k positive integers $T_1, \cdots, T_k$ such that for each $j \in [k]$, $T_j \geq w(t_j)$ and $\sum_{i=1}^{k} T_i = w(V)$, there exists a k-connected-partition of V into $\cup_{j=1}^{k} V_j$, such that for each $j \in [k]$, $t_j \in V_j$ and $w(V_j) \leq T_j + \max_{v \in V} w(v) - 1$.*

One of our main contributions is to give the first elementary and constructive proof by providing a *local search* algorithm with running time $\mathcal{O}^*(4^n)$.

▶ **Theorem 3.** *(a) There is an algorithm which given a k-connected graph, computes a k-connected-partition satisfying the conditions stated in Theorem 2 in time $\mathcal{O}^*(4^n)$.*
(b) If we need a $(\lfloor k/2 \rfloor + 1)$-partition instead of k-partition (the input graph remains assumed to be k-connected), the algorithm's running time improves to $\mathcal{O}^*(2^{\mathcal{O}((n/k)\log k)})$.

We make three remarks. First, the $\mathcal{O}^*(2^{\mathcal{O}((n/k)\log k)})$-time algorithm is a key ingredient of our algorithm for computing a spanning tree with congestion $\mathcal{O}(\sqrt{mn})$. Second, since Theorem 2 guarantees the existence of such a partition, the problem of computing such a partition is not a *decision problem* but a *search problem*. Our algorithm shows that this problem is in the complexity class PLS [12]; we raise its completeness in PLS as an open problem. Third, the running times do not depend on the weights.

The STC Problem, Related Problems and Our Results. Given a connected graph $G = (V, E)$, let T be a spanning tree. For an edge $e = (u, v) \in E$, its detour with respect to T is the unique path from u to v in T; let $\mathsf{DT}(e, T)$ denote the set of edges in this detour. The stretch of e with respect to T is $|\mathsf{DT}(e, T)|$, the length of its detour. The dilation of T is $\max_{e\in E} |\mathsf{DT}(e, T)|$. The edge-congestion of an edge $e \in T$ is $\mathsf{ec}(e, T) := |\{f \in E : e \in \mathsf{DT}(f, T)\}|$, i.e., the number of edges in E whose detours contain e. The congestion of T is $\mathsf{cong}(T) := \max_{e\in T} \mathsf{ec}(e, T)$. The spanning tree congestion (STC) of the graph G is $\mathsf{STC}(G) := \min_T \mathsf{cong}(T)$, where T runs over all spanning trees of G.

We note that there is an equivalent cut-based definition for edge-congestion, which we will use in our proofs. Given a connected graph $G = (V, E)$, an edge set $F \subseteq E$ and two disjoint vertex subsets $V_1, V_2 \subset V$, we let $F(V_1, V_2) := \{ e = \{v_1, v_2\} \in F \mid v_1 \in V_1 \text{ and } v_2 \in V_2 \}$. For each tree-edge $e \in T$, removing e from T results in two connected components; let U_e denote one of the components. Then $\mathsf{ec}(e, T) := |E(U_e, V \setminus U_e)|$.

Various types of congestion, stretch and dilation problems are studied in computer science and discrete mathematics; see the survey [24] for more details. In these problems, one typically seeks a spanning tree (or some other structure) with minimum congestion, stretch or dilation. Among them, the most famous one is the Low Stretch Spanning Tree (LSST) problem, which seeks a spanning tree which minimizes the total stretch of all the edges of G [3]. It is easy to see that minimizing the total stretch is equivalent to minimizing the total edge-congestion of the selected spanning tree. Several strong results were published about the LSST problem. Alon et al. [3] had shown a lower bound of $\Omega(\max\{n \log n, m\})$. Upper bounds have been derived and many efficient algorithms have been devised; the current best upper bound is $\tilde{\mathcal{O}}(m \log n)$. [3, 9, 1, 14, 2] Since total stretch is identical to total edge-congestion, the best upper bound for the LSST problem automatically implies an $\tilde{\mathcal{O}}(\frac{m}{n} \log n)$ upper bound on the *average* edge-congestion. But in the STC problem, we concern the *maximum* edge-congestion.

In comparison, there were not many strong and general results for the STC Problem, though it was studied extensively in the past 13 years. The problem was formally proposed by Ostrovskii [21] in 2004. Prior to this, Simonson [25] had studied the same parameter under a different name to approximate the cut width of outer-planar graph. A number of graph-theoretic results were presented on this topic [22, 17, 16, 15, 7]. Some complexity results were also presented recently [20, 6], but most of these results concern special classes of graphs. The most general result regarding STC of general graphs is an $\mathcal{O}(n\sqrt{n})$ upper bound by Löwenstein, Rautenbach and Regen in 2009 [19], and a matching lower bound by Ostrovskii in 2004 [21]. Note that the above upper bound is not interesting when the graph is sparse, since there is also a trivial upper bound of m. In this paper we come up with a strong improvement to these bounds after 8 years:

Theorem (informal): *For a connected graph G with n vertices and m edges, its spanning tree congestion is at most $\mathcal{O}(\sqrt{mn})$. In terms of average degree $d_{avg} = 2m/n$, we can state this upper bound as $\mathcal{O}(n\sqrt{d_{avg}})$. There is a matching lower bound.*

Our proof for achieving the $\mathcal{O}(\sqrt{mn})$ upper bound is constructive. It runs in exponential time in general; for graphs with $m = \omega(n \log^2 n)$ edges, it runs in sub-exponential time. By using an algorithm of Chen et al. [8] for computing *single-commodity confluent flow* from single-commodity splittable flow, we improve the running time to polynomial, but with a slightly worse upper bound guarantee of $\mathcal{O}(\sqrt{mn} \cdot \log n)$.

Motivated by an open problem raised by Ostrovskii [23] concerning STC of random graphs, we formulate a set of *expanding properties*, and prove that for any graph satisfying these properties, its STC is at most $\mathcal{O}(n)$. We devise a polynomial time algorithm for computing a spanning tree with congestion $\mathcal{O}(n)$ for such graphs. This result, together with a separate lower-bound argument, permit us to show that for random graph $\mathcal{G}(n,p)$ with $1 \geq p \geq \frac{c \log n}{n}$ for some small constant $c > 1$, its STC is $\Theta(n)$ with high probability, thus resolving the open problem raised by Ostrovskii completely.

Min-Max Graph Partitioning and the Generalized Győri-Lovász Theorem. It looks clear that the powerful Theorem 2 can make an impact on graph partitioning. We discuss a number of its consequences which might be of wider interest.

Graph partitioning/clustering is a prominent topic in graph theory/algorithms, and has a wide range of applications. A popular goal is to partition the vertices into sets such that the number of edges across different sets is *small*. While the *min-sum* objective, i.e., minimizing the total number of edges across different sets, is more widely studied, in various applications, the more natural objective is the *min-max* objective, i.e., minimizing the maximum number of edges leaving each set. The min-max objective is our focus here.

Depending on applications, there are additional constraints on the sets in the partition. Two natural constraints are (i) balancedness: the sets are (approximately) balanced in sizes, and (ii) induced-connectivity: each set induces a connected subgraph. The balancedness constraint appears in the application of *domain decomposition* in parallel computing, while the induced-connectivity constraint is motivated by divide-and-conquer algorithms for spanning tree construction. Imposing both constraints simultaneously is not feasible for every graph; for instance, consider the star graph with more than 6 vertices and one wants a 3-partition. Thus, it is natural to ask, for which graphs do partitions satisfying both constraints exist. Theorem 2 implies a simple sufficient condition for existence of such partitions.

By setting the weight of each vertex in G to be its degree, and using the fact that (the maximum degree) $\leq n \leq \frac{2m}{k}$ for any k-connected graph on n vertices and m edges, we have

▶ **Proposition 4.** *If G is a k-connected graph with m edges, then there exists a k-connected-partition, such that the total degree of vertices in each part is at most $4m/k$. Consequently, the min-max objective is also at most $4m/k$.*

Due to expander graphs, this bound is optimal up to a small constant factor. This proposition (together with Lemma 9) implies the following crucial lemma for achieving some of our results.

▶ **Lemma 5.** *Let G be a k-connected graph with m edges. Then $\mathsf{STC}(G) \leq 4m/k$.*

Proposition 4 can be generalized to include approximate balancedness in terms of number of vertices. By setting the weight of each vertex to be cm/n plus its degree in G, we have

▶ **Proposition 6.** *Given any fixed $c > 0$, if G is a k-connected graph with m edges and n vertices, then there exists a k-connected-partition such that the total degree of vertices in each part is at most $(2c+4)m/k$, and the number of vertices in each part is at most $\frac{2c+4}{c} \cdot \frac{n}{k}$.*

Further Related Work. Concerning STC problem, Okamoto et al. [20] gave an $\mathcal{O}^*(2^n)$ algorithm for computing the exact STC of a graph. For graph partitioning, Kiwi, Spielman and Teng [13] formulated the min-max k-partitioning problem and gave bounds for classes of graphs with *small separators*, which were then improved by Steurer [26]. On the algorithmic side, the min-sum objective has been extensively studied; the min-max objective, while striking as the more natural objective in some applications, has received much less attention. The only algorithmic work on this objective (and its variants) are Svitkina and Tardos [28] and Bansal et al. [4]. None of the above work addresses the induced-connectivity constraint.

The classical version of Győri-Lovász Theorem (i.e., the vertex weights are uniform) was proved independently by Győri [10] and Lovász [18]. Lovász used homology theory and is non-constructive. Győri's proof is elementary and is constructive implicitly, but he did not analyze the running time. Polynomial time algorithms for constructing the k-partition were devised for $k = 2, 3$ [27, 29], but no non-trivial finite-time algorithm was known for general graphs with $k \geq 4$. Recently, Hoyer and Thomas [11] provided a clean presentation of Győri's proof by introducing their own terminology, which we use for our constructive proof.

In the full version, we discuss some other related work and some interesting open problems.

2 Technical Overview

To prove the generalized Győri-Lovász theorem, we follow the same framework of Győri's proof [10], and we borrow terminology from the recent presentation by Hoyer and Thomas [11]. But it should be emphasized that proving our generalized theorem is not straight-forward, since in Győri's proof, at each stage a single vertex is moved from one set to other to make progress, while making sure that the former set remains connected. In our setting, in addition to this we also have to ensure that the weights in the partitions do not exceed the specified limit; and hence any vertex that can be moved from one set to another need *not* be candidate for being transferred. The proof of the theorem is presented in Section 3.

As discussed, a crucial ingredient for our upper bound results is Lemma 5, which is a direct corollary of the generalized Győri-Lovász theorem. The lemma takes care of the highly-connected cases; for other cases we provide a recursive way to construct a low congestion spanning tree. See Section 4 for details. For showing our lower bound for general graphs, the challenge is to maintain high congestion while keeping density small. To achieve this, we combine three expander graphs with *little* overlapping between them, and we further make those overlapped vertices of very high degree. This will force a tree-edge adjacent to the centroid of the tree to have high congestion. See Section 5 for details.

We formulate a set of expanding properties which permit constructing a spanning tree of better congestion guarantee in polynomial time. The basic idea is simple: start with a vertex v of high degree as the root, then try to grow the tree by keep attaching new vertices to it, while keeping the invariant that the subtrees rooted at each of the neighbours of v are roughly balanced in size; each such subtree is called a *branch*. But when trying to grow the tree in a balanced way, we will soon realize that as the tree grows, all the remaining vertices might be adjacent only to a few "heavy" branches. To help the balanced growth, our algorithm identifies a *transferable* vertex in a "heavy" branch, and it and its descendants in the tree can be transferred to a "lighter" branch. Another technique is to use multiple rounds of matching between vertices in the tree and the remaining vertices to attach new vertices to the tree. This will tend to make sure that all subtrees do not grow uncontrolled. By showing that random graph satisfies the expanding properties with appropriate parameters, we show that its STC is $\Theta(n)$ with high probability. These results are formally presented in Section 6; see the full version for the complete algorithm and its analysis.

3 Generalized Győri-Lovász Theorem

Let $G(V, E)$ be a k-connected graph on n vertices and m edges, and $w : V \to \mathbb{R}^+$ be a weight function. For any subset $U \subseteq V$, we write $w(U) := \sum_{u \in U} w(u)$. Let $w_{max} := \max_{v \in V} w(v)$. We prove Theorem 3 in this section. Observe that the classical Győri-Lovász Theorem follows from Theorem 3 by taking $w(v) = 1$ for all $v \in V$ and $T_j = n_j$ for all $j \in [k]$. We note that a perfect generalization where one requires that $w(V_j) = T_j$ is not possible – think when all vertex weights are even integers, while some T_j is odd.

3.1 Key Combinatorial Notions

We first highlight the key combinatorial notions used for proving Theorem 3.

Fitted Partial Partition. First, we introduce the notion of *fitted partial partition* (FPP). An FPP A is a tuple of k subsets of V, $(A_1, \ldots, A_k)$, such that the k subsets are pairwise disjoint, and for each $j \in [k]$:

1. $t_j \in A_j$ and $G[A_j]$ is connected, and
2. $w(A_j) \leq T_j + w_{max} - 1$ (we say the set is *fitted* for satisfying this inequality).

We say an FPP is a *Strict Fitted Partial Partition* (SFPP) if $A_1 \cup \cdots \cup A_k$ is a proper subset of V. We say the set A_j is *light* if $w(A_j) < T_j$, and we say it is *heavy* otherwise. Note that there exists at least one light set in any SFPP, for otherwise $w(A_1 \cup \cdots \cup A_k) \geq \sum_{j=1}^{k} T_j = w(V)$, which means $A_1 \cup \cdots \cup A_k = V$. Also note that by taking $A_j = \{t_j\}$, we have an FPP, and hence at least one FPP exists.

Configuration. For a set A_j in an FPP A and a vertex $v \in A_j \setminus \{t_j\}$, we define the reservoir of v with respect to A, denoted by $R_A(v)$, as the vertices in the same connected component as t_j in $G[A_j] \setminus \{v\}$. Note that $v \notin R_A(v)$.

For a *heavy* set A_j, a sequence of vertices $(z_1, \ldots, z_p)$ for some $p \geq 0$ is called a cascade of A_j if $z_1 \in A_j \setminus \{t_j\}$ and $z_{i+1} \in A_j \setminus R_A(z_i)$ for all $1 \leq i < p$. The cascade is called a null cascade if $p = 0$, i.e., if the cascade is empty. Note that for light set, we do *not* need to define its cascade since we do not use it in the proof.

A configuration $\mathcal{C}_A$ is defined as a pair (A, D), where $A = (A_1, \cdots, A_k)$ is an FPP, and D is a set of cascades, which consists of exactly one cascade (possibly, a null cascade) for each heavy set in A. A vertex in some cascade of the configuration is called a cascade vertex.

Given a configuration, we define rank and level inductively as follows. Any vertex in a light set is said to have level 0. For $i \geq 0$, a cascade vertex is said to have rank $i + 1$ if it has an edge to a level-i vertex but does not have an edge to any level-i' vertex for $i' < i$. A vertex u is said to have level i, for $i \geq 1$, if $u \in R_A(v)$ for some rank-i cascade vertex v, but $u \notin R_A(w)$ for any cascade vertex w such that rank of w is less than i. A vertex that is not in $R_A(v)$ for any cascade vertex v is said to have level ∞.

A configuration is called a valid configuration if for each heavy set A_j, rank is defined for each of its cascade vertices and the rank is strictly increasing in the cascade, i.e., if $\{z_1, \ldots, z_p\}$ is the cascade, then $\mathsf{rank}(z_1) < \cdots < \mathsf{rank}(z_p)$. Note that by taking $A_j = \{t_j\}$ and taking the null cascade for each heavy set (in this case A_j is heavy if $w(t_j) = T_j$), we get a valid configuration.

Configuration Vectors and Their Total Ordering. For any vertex, we define its neighborhood level as the smallest level of any vertex adjacent to it. A vertex v of level ℓ is said to satisfy maximality property if each vertex adjacent on it is either a rank-$(\ell+1)$ cascade vertex, has a level of at most $\ell+1$, or is one of the terminals t_j for some j. For any $\ell \geq 0$, a valid configuration is called an ℓ-maximal configuration if all vertices having level at most $\ell-1$ satisfy the maximality property. Note that by definition, any valid configuration is a 0-maximal configuration.

For a configuration $\mathcal{C}_A = ((A_1, \ldots, A_k), D)$, we define $S_A := V \setminus (A_1 \cup \cdots \cup A_k)$. An edge uv is said to be a bridge in $\mathcal{C}_A$ if $u \in S_A$, $v \in A_j$ for some $j \in [k]$, and $level(v) \neq \infty$.

A valid configuration $\mathcal{C}_A$ is said to be ℓ-good if the highest rank of a cascade vertex in $\mathcal{C}_A$ is exactly ℓ (if there are no cascade vertices, then we take the highest rank as 0), $\mathcal{C}_A$ is ℓ-maximal, and all bridges uv in $\mathcal{C}_A$ (if any) are such that $u \in S_A$ and $level(v) = \ell$. Note that taking $A_j = \{t_j\}$ and taking the null cascade for each heavy set gives a 0-good configuration.

For each configuration $\mathcal{C}_A = (A, D)$, we define a configuration vector as below:

$$(L_A \ , \ N_A^0 \ , \ N_A^1 \ , \ N_A^2 \ , \ \ldots \ , \ N_A^n \ ),$$

where L_A is the number of light sets in A, and N_A^ℓ is the number of all level-ℓ vertices in $\mathcal{C}_A$.

Next, we define ordering on configuration vectors. Let $\mathcal{C}_A$ and $\mathcal{C}_B$ be configurations. We say $\mathcal{C}_A >_0 \mathcal{C}_B$ if

- $L_A < L_B$, or
- $L_A = L_B$, and $N_A^0 > N_B^0$.

We say $\mathcal{C}_A =_0 \mathcal{C}_B$ if $L_A = L_B$ and $N_A^0 = N_B^0$. We say $\mathcal{C}_A \geq_0 \mathcal{C}_B$ if $\mathcal{C}_A =_0 \mathcal{C}_B$ or $\mathcal{C}_A >_0 \mathcal{C}_B$. We say $\mathcal{C}_A =_\ell \mathcal{C}_B$ if $L_A = L_B$, and $N_A^{\ell'} = N_B^{\ell'}$ for all $\ell' \leq \ell$.

For $1 \leq \ell \leq n$, we say $\mathcal{C}_A >_\ell \mathcal{C}_B$ if

- $\mathcal{C}_A >_{\ell-1} \mathcal{C}_B$, or
- $\mathcal{C}_A =_{\ell-1} \mathcal{C}_B$, and $N_A^\ell > N_B^\ell$.

We say $\mathcal{C}_A \geq_\ell \mathcal{C}_B$ if $\mathcal{C}_A =_\ell \mathcal{C}_B$ or $\mathcal{C}_A >_\ell \mathcal{C}_B$. We say $\mathcal{C}_A > \mathcal{C}_B$ ($\mathcal{C}_A$ is *strictly better* than $\mathcal{C}_B$) if $\mathcal{C}_A >_n \mathcal{C}_B$.

3.2 Proof of Theorem 3(a)

We use two technical lemmas about configuration vectors and their orderings to prove Theorem 3(a). The proof of Theorem 3(b) follows closely with the proof of Theorem 3(a), but makes use of an observation to give an improved bound on the number of configuration vectors navigated by the local search algorithm.

▶ **Lemma 7.** *Given any ℓ-good configuration $\mathcal{C}_A = (A = (A_1, \ldots, A_k), D_A))$ that does not have a bridge, we can find an $(\ell+1)$-good configuration $\mathcal{C}_B = (B = (B_1, B_2, \ldots, B_k), D_B)$ in polynomial time such that $\mathcal{C}_B > \mathcal{C}_A$.*

▶ **Lemma 8.** *Given an ℓ-good configuration $\mathcal{C}_A = (A = (A_1, \ldots, A_k), D_A)$ having a bridge, we can find in polynomial time a valid configuration $\mathcal{C}_B = (B = (B_1, \ldots, B_k), D_B)$ such that one of the following holds:*

- *$\mathcal{C}_B >_\ell \mathcal{C}_A$, and $\mathcal{C}_B$ is an ℓ-good configuration, or*
- *$\mathcal{C}_B \geq_{\ell-1} \mathcal{C}_A$, there is a bridge $u'v'$ in $\mathcal{C}_B$ such that $u' \in S_B$ and $level(v') \leq \ell - 1$, and $\mathcal{C}_B$ is an $(\ell-1)$-good configuration.*

Proof of Theorem 3(a): We always maintain a configuration $\mathcal{C}_A = (A, D_A)$ that is ℓ-good for some $\ell \geq 0$. If the FPP A is not an SFPP at any point, then we are done. So assume A is an SFPP.

We start with the 0-good configuration where $A_j = \{t_j\}$ and the cascades of all heavy sets are null cascades. If our current configuration $\mathcal{C}_A$ is an ℓ-good configuration that has no bridge, then we use Lemma 7 to get a configuration $\mathcal{C}_B$ such that $\mathcal{C}_B > \mathcal{C}_A$ and B is $(\ell+1)$-good. We take $\mathcal{C}_B$ as the new current configuration $\mathcal{C}_A$. If our current configuration $\mathcal{C}_A$ is an ℓ-good configuration with a bridge, then we get an ℓ'-good configuration $\mathcal{C}_B$ for some $\ell' \geq 0$ such that $\mathcal{C}_B > \mathcal{C}_A$ by repeatedly applying Lemma 8 at most ℓ times. So in either case, we get a strictly better configuration that is ℓ'-good for some $\ell' \geq 0$ in polynomial time. We call this an iteration of our algorithm.

Notice that the number of iterations possible is at most the number of distinct configuration vectors possible. It is easy to see that the number of distinct configuration vectors with highest rank at most r is at most $\binom{n+r-1}{n}$. Since rank of any point is at most n, the number of iterations of our algorithm is at most $(k+1) \cdot \binom{2n}{n} \leq n \cdot 4^n$. Since each iteration runs in polynomial time as guaranteed by the two lemmas, the required running time is $\mathcal{O}^*(4^n)$.

When the algorithm terminates, the FPP given by the current configuration is not an SFPP and this gives the required partition.

Proof of Lemma 7: Since $\mathcal{C}_A$ is ℓ-maximal, any vertex that is at level $\ell' < \ell$ satisfies maximality property. So, for satisfying $(\ell+1)$-maximality, we only need to worry about the vertices that are at level ℓ. Let X_j be the set of all vertices $x \in A_j$ such that x is adjacent to a level-ℓ vertex, $level(x) \geq \ell + 1$ (i.e., $level(x) = \infty$ as the highest rank of any cascade vertex is ℓ), $x \neq t_j$, and x is not a cascade vertex of rank ℓ.

We claim that there exists at least one j for which X_j is not empty. If that is not the case, then we exhibit a cut set of size at most $k-1$. For each j such that A_j is a heavy set with a non-null cascade, let y_j be the highest ranked cascade vertex in A_j. For each j such that A_j is a heavy set with a null cascade, let y_j be t_j. Let Y be the set of all y_j such that A_j is a heavy set. Note that $|Y| \leq k-1$ as A is an SFPP and hence has at least one light set. Let Z_∞ be the set of all vertices in $V \setminus Y$ that have level ∞ and Z be the remaining vertices in $V \setminus Y$. Since A is an SFPP, $S_A \neq \emptyset$, and since all vertices in S_A have level ∞, we have that $Z_\infty \neq \emptyset$. Z is not empty because there exists at least one light set in A and the vertices in a light set have level 0. We show that there is no edge between Z_∞ and Z in G. Suppose there exists an edge uv such that $u \in Z_\infty$ and $v \in Z$. If $u \in S_A$, then uv is a bridge which is a contradiction by our assumption that $\mathcal{C}_A$ does not have a bridge. Hence $u \in A_j$ for some $j \in [k]$. Note that A_j has to be a heavy set, otherwise u has level 0. We have that u is not a cascade vertex (as all cascade vertices with level ∞ are in Y) and $u \neq t_j$ (as all t_j such that $level(t_j) = \infty$ are in Y). Also, v is not of level ℓ as otherwise, $u \in X_j$ but we assumed X_j is empty. But then, v has level at most $\ell - 1$, u has level ∞, and there is an edge uv. This means that $\mathcal{C}_A$ was not ℓ-maximal, which is a contradiction. Thus, there exists at least one j for which X_j is not empty.

For any j such that $X_j \neq \emptyset$, there is at least one vertex x_j such that $X_j \setminus \{x_j\} \subseteq R_A(x_j)$. Now we give the configuration $\mathcal{C}_B$ as follows. We set $B_j = A_j$ for all $j \in [k]$. For each heavy set A_j such that $X_j \neq \emptyset$, we take the cascade of B_j as the cascade of A_j appended with x_j. For each heavy set A_j such that $X_j = \emptyset$, we take the cascade of B_j as the cascade of A_j. It is easy to see that $\mathcal{C}_B$ is $(\ell+1)$-maximal as each vertex that had an edge to level-ℓ vertices in $\mathcal{C}_A$ is now either a rank $\ell+1$ cascade vertex or a level-$(\ell+1)$ vertex or is t_j for some j.

Also, notice that all the new cascade vertices that we introduce (i.e., the x_j's) have their rank as $\ell+1$ and there is at least one rank $\ell+1$ cascade vertex as X_j is not empty for some j. Since there were no bridges in $\mathcal{C}_A$, all bridges in $\mathcal{C}_B$ has to be from S_B to a vertex having level $\ell+1$. Hence, $\mathcal{C}_B$ is $(\ell+1)$-good. All vertices that had level at most ℓ in $\mathcal{C}_A$ retained their levels in $\mathcal{C}_B$. And, at least one level-∞ vertex of $\mathcal{C}_A$ became a level-$(\ell+1)$ vertex in $\mathcal{C}_B$ because the cascade vertex that was at rank ℓ becomes level-$(\ell+1)$ vertex now in at least one set. Since $\mathcal{C}_A$ had no level-$(\ell+1)$ vertices, this means that $\mathcal{C}_B > \mathcal{C}_A$.

Proof of Lemma 8: Let uv be a bridge where $u \in S_A$. Let A_{j^*} be the set containing v. Note that $level(v) = \ell$ because $\mathcal{C}_A$ is ℓ-good. We keep $B_j = A_j$ for all $j \neq j^*$. But we modify A_{j^*} to get B_{j^*} as described below. We maintain that if A_j is a heavy set then B_j is also a heavy set for all j, and hence maintain that $L_B \leq L_A$.

Case 1: A_{j^*} is a light set (i.e., when $\ell = 0$). We take $B_{j^*} = A_{j^*} \cup \{u\}$. For all j such that B_j is a heavy set, cascade of B_j is taken as the null cascade. We have $w(A_{j^*}) \leq T_j - 1$ because A_{j^*} is a light set. So, $w(B_{j^*}) = w(A_{j^*}) + w(u) \leq (T_j - 1) + w_{max}$, and hence B_{j^*} is fitted. Also, $G[B_{j^*}]$ is connected and hence $(B_1, \ldots, B_k)$ is an FPP. We have $\mathcal{C}_B >_0 \mathcal{C}_A$ because either B_{j^*} became a heavy set in which case $L_B < L_A$, or it is a light set in which case $L_B = L_A$ and $N^0_B > N^0_A$. It is easy to see that $\mathcal{C}_B$ is 0-good.

Case 2: A_{j^*} is a heavy set i.e., when $\ell \geq 1$.

Case 2.1: $w(A_{j^*} \cup \{u\}) \leq T_j + w_{max} - 1$. We take $B_{j^*} = A_{j^*} \cup \{u\}$. For each j such that B_j is a heavy set (A_j is also heavy set for such j), the cascade of B_j is taken as the cascade of A_j. $G[B_{j^*}]$ is clearly connected and B_{j^*} is fitted by assumption of the case that we are in. Hence B is indeed an FPP. Observe that all vertices that had level $\ell' \leq \ell$ in $\mathcal{C}_A$ still has level ℓ' in $\mathcal{C}_B$. Since $level(v)$ was ℓ in $\mathcal{C}_A$ by ℓ-goodness of $\mathcal{C}_A$, u also has level ℓ in $\mathcal{C}_B$; and u had level ∞ in $\mathcal{C}_A$. Hence, $\mathcal{C}_B >_\ell \mathcal{C}_A$. It is also easy to see that $\mathcal{C}_B$ remains ℓ-good.

Case 2.2: $w(A_{j^*} \cup \{u\}) \geq T_j + w_{max}$. Let z be the cascade vertex of rank ℓ in A_{j^*}. Note that A_{j^*} should have such a cascade vertex as $v \in A_{j^*}$ has level ℓ. Let $\bar{R}$ be $A_{j^*} \setminus (R_A(z) \cup z)$, i.e., $\bar{R}$ is the set of all vertices in $A_{j^*} \setminus \{z\}$ with level ∞. We initialize $B_{j^*} := A_{j^*} \cup \{u\}$. Now, we delete vertices one by one from B_{j^*} in a specific order until B_{j^*} becomes fitted. We choose the order of deleting vertices such that $G[B_{j^*}]$ remains connected. Consider a spanning tree τ of $G[\bar{R} \cup \{z\}]$. τ has at least one leaf, which is not z. We delete this leaf from B_{j^*} and τ. We repeat this process until τ is just the single vertex z or B_{j^*} becomes fitted. If B_{j^*} is not fitted even when τ is the single vertex z, then delete z from B_{j^*}. If B_{j^*} is still not fitted then delete u from B_{j^*}. Note that at this point $B_{j^*} \subset A_{j^*}$ and hence is fitted. Also, note that $G[B_{j^*}]$ remains connected. Hence $(B_1, \ldots, B_k)$ is an FPP. B_{j^*} does not become a light set because B_j became fitted when the last vertex was deleted from it. Before this vertex was deleted, it was not fitted and hence had weight at least $T_{j^*} + w_{max}$ before this deletion. Since the last vertex deleted has weight at most w_{max}, B_{j^*} has weight at least T_{j^*} and hence is a heavy set. Now we branch into two subcases for defining the cascades.

Case 2.2.1: $z \in B_{j*}$ (i.e, z was not deleted from B_{j^*} in the process above). For each j such that B_j is a heavy set, the cascade of B_j is taken as the cascade of A_j. Since a new ℓ level vertex u is added and all vertices that had level at most ℓ retain their level, we have that $\mathcal{C}_B >_\ell \mathcal{C}_A$. It is also easy to see that $\mathcal{C}_B$ remains ℓ-good.

Case 2.2.2: $z \notin B_{j^*}$ (i.e, z was deleted from B_{j^*}). For each j such that B_j is a heavy set, the cascade of B_j is taken as the cascade of A_j but with the rank ℓ cascade vertex (if it has any) deleted from it. $\mathcal{C}_B \geq_{\ell-1} \mathcal{C}_A$ because all vertices that were at a level of $\ell' = \ell - 1$ or

smaller, retain their levels. Observe that there are no bridges in $\mathcal{C}_B$ to vertices that are at a level at most $\ell-2$, all vertices at a level at most $\ell-2$ still maintain the maximality property, and we did not introduce any cascade vertices. Hence, $\mathcal{C}_B$ is $(\ell-1)$-good. It only remains to prove that there is a bridge $u'v'$ in $\mathcal{C}_B$ such that $level(v') \leq \ell-1$. We know $z \in S_B$. Since z was a rank ℓ cascade vertex in $\mathcal{C}_A$, z had an edge to z' such that z' had level $\ell-1$ in $\mathcal{C}_A$. Observe that level of z' is at most $\ell-1$ in $\mathcal{C}_B$ as well. Hence, taking $u'v' = zz'$ completes the proof.

4 Upper Bounds for Spanning Tree Congestion

We first state the following easy lemma, which together with Proposition 4, implies Lemma 5.

▶ **Lemma 9.** *In a graph $G=(V,E)$, let t_1 be a vertex, and let $t_2,\cdots,t_\ell$ be any $(\ell-1)$ neighbours of t_1. Suppose that there exists a ℓ-connected-partition $\cup_{j=1}^{\ell} V_\ell$ such that for all $j \in \ell$, $t_j \in V_j$, and the sum of degree of vertices in each V_j is at most D. Let τ_j be an arbitrary spanning tree of $G[V_j]$. Let e_j denote the edge $\{t_1,t_j\}$. Let τ be the spanning tree of G defined as $\tau := \left(\cup_{j=1}^{\ell}\ \tau_j\right) \bigcup \left(\cup_{j=2}^{\ell}\ e_j\right)$. Then τ has congestion at most D.*

▶ **Theorem 10.** *For any connected graph $G=(V,E)$, there is an algorithm which computes a spanning tree with congestion at most $8\sqrt{mn}$ in time $\mathcal{O}^*\left(2^{\mathcal{O}\left(n\log n/\sqrt{m/n}\right)}\right)$.*

▶ **Theorem 11.** *For any connected graph $G=(V,E)$, there is a polynomial time algorithm which computes a spanning tree with congestion at most $16\sqrt{mn}\log n$.*

The two algorithms follows the same framework, depicted in Algorithm 1. It is a recursive algorithm; the parameter $\hat{m}$ is a global parameter, which is the number of edges in the input graph G in the first level of the recursion; let $\hat{n}$ denote the number of vertices in this graph.

The only difference between the two algorithms is in Line 15 on how this step is executed, with trade-off between the running time of the step $T(\hat{m}, n_{_H}, m_{_H})$, and the guarantee $D(\hat{m}, n_{_H}, m_{_H})$. For proving Theorem 10, we use Theorem 3(b), Proposition 4 and Lemma 9, yielding $D(\hat{m}, n_{_H}, m_{_H}) \leq 8m_{_H}\sqrt{n_{_H}/\hat{m}}$ and $T(\hat{m}, n_{_H}, m_{_H}) = \mathcal{O}^*\left(2^{\mathcal{O}(n_{_H}\log n_{_H}/\sqrt{\hat{m}/n_{_H}})}\right)$. For proving Theorem 11, we make use of an algorithm in Chen et al. [8], which yields $D(\hat{m}, n_{_H}, m_{_H}) \leq 16m_{_H}\sqrt{n_{_H}/\hat{m}}\log n_{_H}$ and $T(\hat{m}, n_{_H}, m_{_H}) = \mathsf{poly}(n_{_H}, m_{_H})$. Next, we discuss the algorithm in Chen et al., then we prove Theorem 11.

Single-Commodity Confluent Flow. In a *single-commodity confluent flow* problem, the input includes a graph $G=(V,E)$, a *demand* function $w: V \to \mathbb{R}^+$ and ℓ sinks $t_1,\cdots,t_\ell \in V$. For each $v \in V$, a flow of amount $w(v)$ is routed from v to one of the sinks. But there is a restriction: at every vertex $u \in V$, the outgoing flow must leave u on at most 1 edge, i.e., the outgoing flow from u is unsplittable. The problem is to seek a flow satisfying the demands which minimizes the *node congestion*, i.e., the maximum incoming flow among all vertices. Since the incoming flow is maximum at one of the sinks, it is equivalent to minimize the maximum flow received among all sinks. (We assume that no flow entering a sink will leave.) *Single-commodity splittable flow* problem is almost identical to single-commodity confluent flow problem, except that the above restriction is dropped, i.e., now the outgoing flow at u can split along multiple edges. Note that here, the maximum incoming flow might not be at a sink. It is known that single-commodity splittable flow can be solved in polynomial time. For brevity, we drop the phrase "single-commodity" from now on. Corollary 13 below follows from Theorem 12 and Proposition 4.

Algorithm 1: `FindLCST`$(H, \hat{m})$

Input : A connected graph $H = (V_H, E_H)$ on n_H vertices and m_H edges
Output : A spanning tree τ of H

1 **if** $m_H \leq 8\sqrt{\hat{m} n_H}$ **then**
2 | **return** an arbitrary spanning tree of H
3 **end**
4 $k \leftarrow \left\lceil \sqrt{\hat{m}/n_H} \right\rceil$
5 $Y \leftarrow$ a global minimum vertex cut of H
6 **if** $|Y| < k$ **then**
7 | $X \leftarrow$ the smallest connected component in $H[V_H \setminus Y]$
8 | $Z \leftarrow V_H \setminus (X \cup Y)$
9 | $\tau_1 \leftarrow$ `FindLCST`$(H[X], \hat{m})$
10 | $\tau_2 \leftarrow$ `FindLCST`$(H[Y \cup Z], \hat{m})$; ($H[Y \cup Z]$ is connected as Y is a global min cut)
11 | **return** $\tau_1 \cup \tau_2 \cup$ (an arbitrary edge between X and Y)
12 **else**
13 | $t_1 \leftarrow$ an arbitrary vertex in V_H
14 | Pick $\lfloor k/2 \rfloor$ neighbors of t_1 in the graph H; denote them by $t_2, t_3, \cdots, t_{\lfloor k/2 \rfloor + 1}$. Let e_j denote edge $t_1 t_j$ for $2 \leq j \leq \lfloor k/2 \rfloor + 1$.
15 | Compute a $(\lfloor k/2 \rfloor + 1)$-connected-partition of H, denoted by $\cup_{j=1}^{\lfloor k/2 \rfloor + 1} V_j$, such that for each $j \in [\lfloor k/2 \rfloor + 1]$, $t_j \in V_j$, and the total degree (w.r.t. graph H) of vertices in each V_j is at most $D(\hat{m}, n_H, m_H)$. Let the time needed be $T(\hat{m}, n_H, m_H)$.
16 | For each $j \in [\lfloor k/2 \rfloor + 1]$, $\tau_j \leftarrow$ an arbitrary spanning tree of $G[V_j]$
17 | **return** $\left(\cup_{j=1}^{\lfloor k/2 \rfloor + 1} \tau_j\right) \bigcup \left(\cup_{j=2}^{\lfloor k/2 \rfloor + 1} e_j\right)$
18 **end**

▶ **Theorem 12** ([8, Section 4]). *Suppose that given graph G, demand w and ℓ sinks, there is a splittable flow with node congestion q. Then there exists a polynomial time algorithm which computes a confluent flow with node congestion at most $(1 + \ln \ell)q$ for the same input.*

▶ **Corollary 13.** *Let G be a k-connected graph with m edges. Then for any $\ell \leq k$ and for any ℓ vertices $t_1, \cdots, t_\ell \in V$, there exists a polynomial time algorithm which computes an ℓ-connected-partition $\cup_{j=1}^{\ell} V_\ell$ such that for all $j \in \ell$, $t_j \in V_j$, and the total degrees of vertices in each V_j is at most $4(1 + \ln \ell)m/\ell$.*

Congestion Analysis. We view the whole recursion process as a recursion tree. There is no endless loop, since down every path in the recursion tree, the number of vertices in the input graphs are *strictly* decreasing. On the other hand, note that the leaf of the recursion tree is resulted by either (i) when the input graph H to that call satisfies $m_H \leq 8\sqrt{\hat{m} n_H}$, or (ii) when Lines 13–17 are executed. An internal node appears only when the vertex-connectivity of the input graph H is *low*, and it makes two recursion calls.

We prove the following statement by induction from bottom-up: for each graph which is the input to some call in the recursion tree, the returned spanning tree of that call has congestion at most $16\sqrt{\hat{m} n_H} \log n_H$.

We first handle the two basis cases (i) and (ii). In case (i), `FindLCST` returns an arbitrary spanning tree, and the congestion is bounded by $m_H \leq 8\sqrt{\hat{m}n_H}$. In case (ii), by Corollary 13 and Lemma 9, `FindLCST` returns a tree with congestion at most $16m_H\sqrt{n_H/\hat{m}}\log n_H \leq 16\sqrt{\hat{m}n_H}\log n_H$.

Next, let H be the input graph to a call which is represented by an internal node of the recursion tree. Recall the definitions of X, Y, Z, τ_1, τ_2 in the algorithm.

Let $|X| = x$. Note that $1 \leq x \leq n_H/2$. Then by induction hypothesis, the congestion of the returned spanning tree is at most

$$\begin{aligned}&\max\{\text{ congestion of } \tau_1 \text{ in } H[X]\ ,\ \text{congestion of } \tau_2 \text{ in } H[Y \cup Z]\ \} \ +\ |X|\cdot|Y| \\ &\leq 16\sqrt{\hat{m}(n_H - x)}\log(n_H - x) \ +\ \left(\sqrt{\hat{m}/n_H} + 1\right)\cdot x.\end{aligned}$$

Viewing x as a real variable, by taking derivative, it is easy to see that the above expression is maximized at $x = 1$. Thus the congestion is at most $16\sqrt{\hat{m}(n_H - 1)}\log(n_H - 1) + \sqrt{\hat{m}/n_H} + 1 \ \leq\ 16\sqrt{\hat{m}n_H}\log n_H$. In the full version, we do the running time analysis.

5 Lower Bound for Spanning Tree Congestion

Here, we give a lower bound on spanning tree congestion which matches our upper bound.

▶ **Theorem 14.** *For any sufficiently large n, and for any m satisfying $n^2/2 \geq m \geq \max\{16n\log n, 100n\}$, there exists a connected graph with $N = (3 - o(1))n$ vertices and $M \in [m, 7m]$ edges, for which the spanning tree congestion is at least $\Omega(\sqrt{mn})$.*

By some standard random graph arguments, we show that when $n^2/2 \geq m \geq 16n\log n$, there exists a connected graph $H(n, m)$ with n vertices and $[m/2, 2m]$ edges, such that for each subset of vertices S with $|S| \leq n/2$, the number of edges leaving S is $\Omega((m/n)\cdot|S|)$.

We discuss our construction for Theorem 14. The full proof is in the full version. The vertex set V is the union of three vertex subsets V_1, V_2, V_3, such that $|V_1| = |V_2| = |V_3| = n$, $|V_1 \cap V_2| = |V_2 \cap V_3| = \sqrt{m/n}$, and V_1, V_3 are disjoint. In each of V_1, V_2 and V_3, we embed $H(n, m)$. Up to this point, the construction is similar to that of Ostrovskii [21], except that we use $H(n, m)$ instead of a complete graph. The new component in our construction is adding the following edges. For each vertex $v \in V_1 \cap V_2$, add an edge between v and every vertex in $(V_1 \cup V_2) \setminus \{v\}$. Similarly, for each vertex $v \in V_3 \cap V_2$, add an edge between v and every vertex in $(V_3 \cup V_2) \setminus \{v\}$. This new component is crucial: without it, we could only prove a lower bound of $\Omega(m/\sqrt{n}) = \Omega(\sqrt{mn}\cdot\frac{\sqrt{m}}{n})$.

6 Graphs with Expanding Properties

For any vertex subset $U, W \subset V$, let $N_W(U)$ denote the set of vertices in W which are adjacent to a vertex in U. Let $N(U) := N_{V\setminus U}(U)$.

▶ **Definition 15.** A graph $G = (V, E)$ on n vertices is an (n, s, d_1, d_2, d_3, t)-expanding graph if the following four conditions are satisfied:

(1) for each vertex subset S with $|S| = s$, $|N(S)| \geq d_1 n$;
(2) for each vertex subset S with $|S| \leq s$, $|N(S)| \geq d_2|S|$;
(3) for each vertex subset S with $|S| \leq n/2$ and for any subset $S' \subset S$, $|N_{V\setminus S}(S')| \ \geq\ |S'| - t$.
(4) For each vertex subset S, $|E(S, V \setminus S)| \leq d_3|S|$.

▶ **Theorem 16.** *For any connected graph G which is an (n, s, d_1, d_2, d_3, t)-expanding graph, there is a polynomial time algorithm which computes a spanning tree with congestion at most*

$$d_3 \cdot \left[4 \cdot \max \left\{ s+1 \ , \ \left\lceil \frac{3d_1 n}{d_2} \right\rceil \right\} \cdot \left(\frac{1}{2d_1} \right)^{\log_{(2-\delta)} 2} + t \right], \quad \textit{where } \delta = \frac{t}{d_1 n}.$$

In the full version, we show that for random graph $\mathcal{G}(n,p)$ with $p \geq 64 \log n/n$. with high probability the graph is an (n, s, d_1, d_2, d_3, t)-expanding graph with $s = \Theta(1/p)$, $d_1 = \Theta(1)$, $d_2 = \Theta(np)$, $d_3 = \Theta(np)$, $t = \Theta(1/p)$ (and hence $\delta = o(1)$). Applying the above theorem, together with a separate lower bound argument, we show:

▶ **Theorem 17.** *If $G \in \mathcal{G}(n,p)$ where $p \geq 64 \log n/n$, then with probability at least $1 - \mathcal{O}(1/n)$, its STC is $\Theta(n)$.*

References

1. Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. In *FOCS 2008*, pages 781–790, 2008.
2. Ittai Abraham and Ofer Neiman. Using petal-decompositions to build a low stretch spanning tree. In *STOC 2012*, pages 395–406, 2012.
3. Noga Alon, Richard M. Karp, David Peleg, and Douglas B. West. A graph-theoretic game and its application to the k-server problem. *SIAM J. Comput.*, 24(1):78–100, 1995.
4. 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.
5. Sandeep N. Bhatt, Fan R. K. Chung, Frank Thomson Leighton, and Arnold L. Rosenberg. Optimal simulations of tree machines (preliminary version). In *FOCS 1986*, pages 274–282, 1986.
6. Hans L. Bodlaender, Fedor V. Fomin, Petr A. Golovach, Yota Otachi, and Erik Jan van Leeuwen. Parameterized complexity of the spanning tree congestion problem. *Algorithmica*, 64(1):85–111, 2012.
7. Hans L. Bodlaender, Kyohei Kozawa, Takayoshi Matsushima, and Yota Otachi. Spanning tree congestion of k-outerplanar graphs. *Discrete Mathematics*, 311(12):1040–1045, 2011.
8. 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. *J. ACM*, 54(4):16, 2007.
9. Michael Elkin, Yuval Emek, Daniel A. Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. *SIAM J. Comput.*, 38(2):608–628, 2008.
10. E. Győri. On division of graphs to connected subgraphs. *Colloq. Math. Soc. Janos Bolyai*, 18:485–494, 1976.
11. Alexander Hoyer and Robin Thomas. The Győri-Lovász theorem. *arXiv*, abs/1605.01474, 2016. URL: `http://arxiv.org/abs/1706.08115`.
12. David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988.
13. Marcos A. Kiwi, Daniel A. Spielman, and Shang-Hua Teng. Min-max-boundary domain decomposition. *Theor. Comput. Sci.*, 261(2):253–266, 2001.
14. Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly $O(m \log n)$ time solver for SDD linear systems. In *FOCS 2011*, pages 590–598, 2011.
15. Kyohei Kozawa and Yota Otachi. Spanning tree congestion of rook's graphs. *Discussiones Mathematicae Graph Theory*, 31(4):753–761, 2011.

16 Kyohei Kozawa, Yota Otachi, and Koichi Yamazaki. On spanning tree congestion of graphs. *Discrete Mathematics*, 309(13):4215–4224, 2009.

17 Hiu Fai Law, Siu Lam Leung, and Mikhail I. Ostrovskii. Spanning tree congestions of planar graphs. *Involve*, 7(2):205–226, 2014.

18 László Lovász. A homology theory for spanning trees of a graph. *Acta Math. Acad. Sci. Hungaricae*, 30(3–4):241–251, 1977.

19 Christian Löwenstein, Dieter Rautenbach, and Friedrich Regen. On spanning tree congestion. *Discrete Math.*, 309(13):4653–4655, 2009.

20 Yoshio Okamoto, Yota Otachi, Ryuhei Uehara, and Takeaki Uno. Hardness results and an exact exponential algorithm for the spanning tree congestion problem. *J. Graph Algorithms Appl.*, 15(6):727–751, 2011.

21 M. I. Ostrovskii. Minimal congestion trees. *Discrete Math.*, 285:219–226, 2004.

22 M. I. Ostrovskii. Minimum congestion spanning trees in planar graphs. *Discrete Math.*, 310:1204–1209, 2010.

23 M. I. Ostrovskii. Minimum congestion spanning trees in bipartite and random graphs. *Acta Mathematica Scientia*, 31(2):634–640, 2011.

24 André Raspaud, Ondrej Sýkora, and Imrich Vrto. Congestion and dilation, similarities and differences: A survey. In *SIROCCO 2000*, pages 269–280, 2000.

25 Shai Simonson. A variation on the min cut linear arrangement problem. *Mathematical Systems Theory*, 20(4):235–252, 1987.

26 David Steurer. Tight bounds for the min-max boundary decomposition cost of weighted graphs. In *SPAA 2006*, pages 197–206, 2006.

27 Hitoshi Suzuki, Naomi Takahashi, and Takao Nishizeki. A linear algorithm for bipartition of biconnected graphs. *Inf. Process. Lett.*, 33(5):227–231, 1990.

28 Zoya Svitkina and Éva Tardos. Min-max multiway cut. In *APPROX-RANDOM 2004*, pages 207–218, 2004.

29 Koichi Wada and Kimio Kawaguchi. Efficient algorithms for tripartitioning triconnected graphs and 3-edge-connected graphs. In *Graph-Theoretic Concepts in Computer Science, 19th International Workshop, WG '93, Utrecht, The Netherlands, June 16-18, 1993, Proceedings*, pages 132–143, 1993.