Spanning Tree Congestion and Computation of Generalized Győri-Lovász Partition

L. Sunil Chandran [*1], Yun Kuen Cheung [**2], and Davis Issac[2]

[1] Department of Computer Science and Automation, Indian Institute of Science, India.
`sunil@csa.iisc.ernet.in`
[2] Max Planck Institute for Informatics, Saarland Informatics Campus, Germany.
`ycheung@mpi-inf.mpg.de` , `dissac@mpi-inf.mpg.de`

Abstract. 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 root of this problem dates back to at least 30 years ago, motivated by applications in network design, parallel computing and circuit design. Variants of the problem have also seen algorithmic applications as a preprocessing step of several important graph algorithms.

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*, for which Chen et al. [14] gave a non-constructive proof. We give the first elementary and constructive proof by providing a local search algorithm with running time $\mathcal{O}^*(4^n)$, which is a key ingredient of the above-mentioned sub-exponential time algorithm. We discuss a few 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.

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" [8,36], with natural motivations from parallel computing and circuit design applications. The STC problem was formally defined by Ostrovskii [30] in 2004, and since then a number of results have been presented. The probabilistic version of the STC problem, coined as *probabilistic capacity mapping*, also finds applications in several important graph algorithm problems, e.g., the Min-Bisection problem.

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. [14]. Their proof uses advanced techniques in topology and homology theory, and is non-constructive.

Definition 1.1. *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 1.2 ([14, Theorems 25, 26]). *Let $G = (V, E)$ be a k-connected [3] 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*

---

[3] For brevity, we say "k-connected" for "k-vertex-connected" henceforth.

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 $\mathcal{O}^*(4^n)$:[4]

Theorem 1.3. *(a) There is an algorithm which given a k-connected graph, computes a k-connected-partition satisfying the conditions stated in Theorem 1.2 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 1.2 guarantees the existence of such a partition, the problem of computing such a partition is not a *decision problem* but a *search problem*. Our local search algorithm shows that this problem is in the complexity class PLS [20]; 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. 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. In these problems, one typically seeks a spanning tree (or some other structure) with minimum congestion or dilation. We mention some of the well-known problems, where minimization is done over all the spanning trees of the given graph:

1. The Low Stretch Spanning Tree (LSST) problem is to find 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.
2. The STC problem is to find a spanning tree of minimum congestion. [30]

---

[4] $\mathcal{O}^*$ notation hides all polynomial factors in input size.

3. Tree Spanner Problem is to find a spanning tree of minimum dilation. [13] The more general Spanner problem is to find a sparser subgraph of minimum *distortion*. [4]

There are other congestion and dilation problems which do not seek a spanning tree, but some other structure. The most famous among them is the Bandwidth problem and the Cutwidth problem; see the survey [34] for more details.

Among the problems mentioned above, several strong results were published in connection with 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 $\tilde{\mathcal{O}}(m \log n)$. [3,15,1,22,2] 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; as we shall see, for some graphs, the maximum edge-congestion has to be a factor of $\tilde{\Omega}(\sqrt{n^3/m})$ larger than the average 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 [30] in 2004. Prior to this, Simonson [36] 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 [31,25,24,23,10]. Some complexity results were also presented recently [29,9], 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 [27], and a matching lower bound by Ostrovskii in 2004 [30]. 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. [14] 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 [32] 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 $c > 1$,[5] 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 1.2 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 1.2 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 elementary fact that the maximum degree $\Delta(G) \le n \le 2m/k$ for any k-connected graph G on n vertices and m edges, we have

Proposition 1.4. *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 4.1) implies the following crucial lemma for achieving some of our results.

Lemma 1.5. *Let G be a k-connected graph with m edges. Then $\mathsf{STC}(G) \le 4m/k$.*

Proposition 1.4 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 1.6. *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*

---

[5] Note that the STC problem is relevant only for connected graphs. Since the threshold function for graph connectivity is $\frac{\log n}{n}$, this result applies for almost all of the relevant range of values of p.

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. [29] gave an $\mathcal{O}^*(2^n)$ algorithm for computing the exact STC of a graph. The probabilistic version of the STC problem, coined as *probabilistic capacity mapping*, is an important tool for several graph algorithm problems, e.g., the Min-Bisection problem. Räcke [33] showed that in the probabilistic setting, distance and capacity are *interchangeable*, which briefly says a general upper bound for one objective implies the same general upper bound for the other. Thus, due to the above-mentioned results on LSST, there is an upper bound of $\tilde{\mathcal{O}}(\log n)$ on the *maximum average congestion*. Räcke's result also implies an $\mathcal{O}(\log n)$ approximation algorithm to the Min-Bisection problem, improving upon the $\mathcal{O}(\log^{3/2} n)$ approximation algorithm of Feige and Krauthgamer [16]. However, in the deterministic setting, such interchanging phenomenon does not hold: there is a simple tight bound $\Theta(n)$ for dilation, but for congestion it can be as high as $\Theta(n\sqrt{n})$. For the precise definitions, more background and key results about the concepts we have just discussed, we recommend the writing of Andersen and Feige [5].

Graph partitioning/clustering is a prominent research topic with wide applications, so it comes no surprise that a lot of work has been done on various aspects of the topic; we refer readers to the two extensive surveys by Schaeffer [35] and by Teng [41]. Kiwi, Spielman and Teng [21] formulated the min-max k-partitioning problem and gave bounds for classes of graphs with *small separators*, which are then improved by Steurer [38]. On the algorithmic side, many of the related problems are NP-hard, so the focus is on devising approximation algorithms. Sparkled by the seminal work of Arora, Rao and Vazirani [6] on sparsest cut and of Spielman and Teng [37] on local clustering, graph partitioning/clustering algorithms with various constraints have attracted attention across theory and practice; we refer readers to [7] for a fairly recent account of the development. 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 [40] and Bansal et al. [7]. 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 [17] and Lovász [26]. Lovász's proof uses 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 $k = 2, 3$ [39,42], but no non-trivial finite-time algorithm was known for general graphs with $k \geq 4$.[6] Recently, Hoyer and Thomas [19] provided a clean presentation of Győri's proof

---

[6] In 1994, there was a paper by Ma and Ma in *Journal of Computer Science and Technology*, which claimed a poly-time algorithm for all k. However, according to a recent study [18], Ma and Ma's algorithm can fall into an endless loop. Also, Győri said the algorithm should be wrong (see [28]).

by introducing their own terminology, which we use for our constructive proof of Theorem 1.2.

Notation. Given a graph $G = (V, E)$, an edge set $F \subseteq E$ and 2 disjoint vertex subsets $V_1, V_2 \subset V$, we let $F(V_1, V_2) := \{ e = \{v_1, v_2\} \in F \mid v_1 \in V_1 \text{ and } v_2 \in V_2 \}$.

2   Technical Overview

To prove the generalized Győri-Lovász theorem constructively, we follow the same framework of Győri's proof [17], and we borrow terminology from the recent presentation by Hoyer and Thomas [19]. 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 is presented in Section 3.

As discussed, a crucial ingredient for our upper bound results is Lemma 1.5, 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 any spanning 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. Now 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 grow, all the remaining vertices may be seen to be adjacent only to a few number of "heavy" branches. To help the balanced growth, the algorithm will identify a *transferable* vertex which is 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 a random graph has STC of $\Theta(n)$ with high probability.

3   Generalized Győri-Lovász Theorem

We prove Theorem 1.3 in this section. Observe that the classical Győri-Lovász Theorem follows from Theorem 1.2 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.

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 $U \subseteq V$, $w(U) := \sum_{u \in U} w(u)$. Let $w_{max} := \max_{v \in V} w(v)$.

3.1 Key Combinatorial Notions

We first highlight the key combinatorial notions used for proving Theorem 1.3; see Figures 1 and 2 for illustrations of some of these notions.

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. $t_j \in A_j$,
2. $G[A_j]$ is connected and
3. $w(A_j) \leq T_j + w_{max} - 1$ (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 <u>reservoir</u> 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 <u>cascade</u> 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 <u>null cascade</u> 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. (See Figure 1.)

A <u>configuration</u> $\mathcal{C}_A$ 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 that is in some cascade of the configuration is called a <u>cascade vertex</u>.

Given a configuration, we define <u>rank</u> and <u>level</u> 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 <u>valid configuration</u> if for each heavy set A_j, rank is defined for each of its cascade vertices and the rank is strictly increasing in the
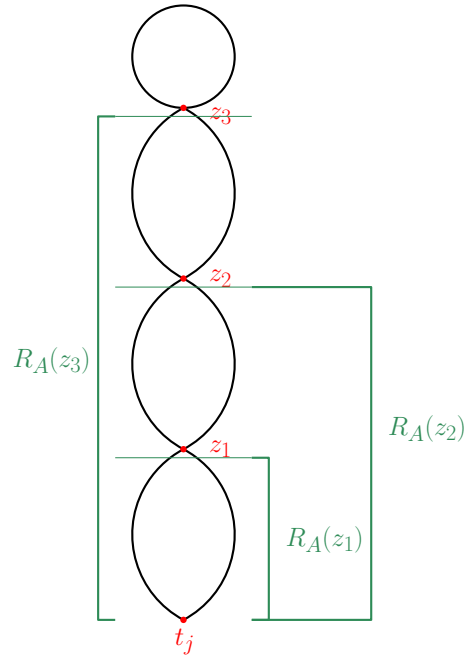
Fig. 1. Given a configuration (A, D) and a heavy set A_j in A, the figure shows a cascade (z_1, z_2, z_3) for the heavy set A_j and several reservoirs of the cascade vertices.
For any z_ℓ, $z_\ell \notin R_A(z_\ell)$. A cascade vertex z_ℓ is a cut-vertex of $G[A_j]$, i.e., $G[A_j \setminus \{z_\ell\}]$ is disconnected. The removal of z_ℓ from A_j will lead to at least two connected components in $G[A_j \setminus \{z_\ell\}]$, and the connected component containing t_j is the reservoir of z_ℓ.
We identify $t_j = z_0$, but we clarify that a terminal vertex is never in a cascade. Each epoch between z_ℓ and $z_{\ell+1}$, and also the epoch above z_3, is a subset of vertices $B \subset A_j$, where $B \ni z_\ell$ and $G[B]$ is connected. Note that in general, it is possible that there is no vertex above the last cascade vertex.

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. (See Figure 2.)

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 $level(v) \neq \infty$.
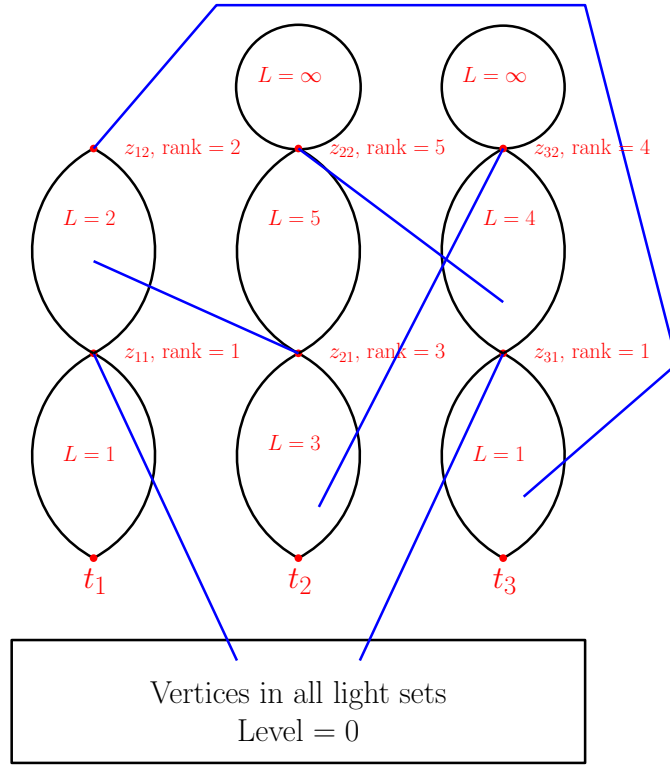
Fig. 2. An instance of a valid configuration. Every blue segment/curve represent an edge from a cascade vertex to a vertex in some reservoir or light set.
Every cascade vertex connected to a light set has rank 1, and all vertices in the epoch immediately below a rank 1 cascade vertex are of level 1. Inductively, every cascade vertex connected to a vertex of level i has rank $i + 1$, and all vertices in the epoch immediately below a rank i cascade vertex are of level i. All vertices above the last cascade vertex of each cascade has level ∞.

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:

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

where L_A is the number of light sets in A, and N_A^ℓ is the total 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 1.3

We use two technical lemmas about configuration vectors and their orderings to prove Theorem 1.3(a). The proof of Theorem 1.3(b) follows closely with the proof of Theorem 1.3(a), but makes use of an observation about the rank of a vertex in the local search algorithm, to give an improved bound on the number of configuration vectors navigated by the algorithm.

Lemma 3.1. *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 $\mathcal{C}_B > \mathcal{C}_A$.*

Proof. 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 $\mathcal{C}_B > \mathcal{C}_A$.   □

Lemma 3.2. *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. 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 $N_B^0 > N_A^0$. 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. $\square$

Proof of Theorem 1.3(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 3.1 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 3.2 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 $(k+1) \cdot \binom{2n}{n}$, which is at most $n \cdot 4^n$. 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 Theorem 1.3(b): . Since any k-connected graph is also $(\lfloor k/2 \rfloor + 1)-$vertex connected, the algorithm will give the required partition due to Theorem 1.3(a). We only need to prove the better running time claimed by Theorem 1.3(b). For this, we show that the highest rank attained by any vertex during the algorithm is at most $2n/(k-2)$. Since the number of distinct configuration vectors with highest rank r is at most $\binom{n+r-1}{n}$, we then have that the running time is $\mathcal{O}^*\binom{n+\frac{2n}{k-2}-1}{n}$, which is $\mathcal{O}^*(2^{\mathcal{O}((n/k)\log k)})$, as claimed. Hence, it only remains to prove that the highest rank is at most $2n/(k-2)$.

For this, observe that in an ℓ-good configuration, for each $0 \leq i < \ell$, the union of all vertices having level i and the set of $(\lfloor k/2 \rfloor + 1)$ terminals together forms a cutset. Since the graph is k-connected, this means that for each $0 \leq i < \ell$, the number of vertices having level i is at least $k/2 - 1$. The required bound on the rank easily follows. □

4   Upper Bounds for Spanning Tree Congestion

We first state the following easy lemma, which together with Proposition 1.4, implies Lemma 1.5.

Lemma 4.1. *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 4.2. *For any connected graph $G = (V, E)$, there is an algorithm which computes a spanning tree with congestion at most $8\sqrt{mn}$ in $\mathcal{O}^* \left(2^{\mathcal{O}\left(n \log n / \sqrt{m/n} \right)} \right)$ time.*

Theorem 4.3. *For any connected graph $G = (V, E)$, there is a polynomial time algorithm which computes a spanning tree with congestion at most $16\sqrt{mn} \cdot \log n$.*

The two algorithms follow 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 4.2, we use Theorem 1.3(b),

Proposition 1.4 and Lemma 4.1, yielding $D(\hat{m}, n_H, m_H) \leq 8m_H \sqrt{n_H/\hat{m}}$ and $T(\hat{m}, n_H, m_H) = \mathcal{O}^* \left(2^{\mathcal{O}\left(n_H \log n_H / \sqrt{\hat{m}/n_H}\right)} \right)$. For proving Theorem 4.3, we make use of an algorithm in Chen et al. [14], which yields $D(\hat{m}, n_H, m_H) \leq 16m_H \sqrt{n_H/\hat{m}} \cdot \log n_H$ and $T(\hat{m}, n_H, m_H) = \mathsf{poly}(n_H, m_H)$.

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 $\quad$ **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 $\quad$ $X \leftarrow$ the smallest connected component in $H[V_H \setminus Y]$ (See Figure 3)
8 $\quad$ $Z \leftarrow V_H \setminus (X \cup Y)$
9 $\quad$ $\tau_1 \leftarrow$ FindLCST$(H[X], \hat{m})$
10 $\quad$ $\tau_2 \leftarrow$ FindLCST$(H[Y \cup Z], \hat{m})$; $\ (H[Y \cup Z]$ is connected as Y is a global min cut)
11 $\quad$ **return** $\tau_1 \cup \tau_2 \cup$ (an arbitrary edge between X and Y)
12 else
13 $\quad$ $t_1 \leftarrow$ an arbitrary vertex in V_H
14 $\quad$ Pick $\lfloor k/2 \rfloor$ neighbours 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 $2 \leq j \leq \lfloor k/2 \rfloor + 1$. (See Figure 4)
15 $\quad$ 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 $\quad$ For each $j \in [\lfloor k/2 \rfloor + 1]$, $\tau_j \leftarrow$ an arbitrary spanning tree of $G[V_j]$
17 $\quad$ **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**

In the rest of this section, we first discuss the algorithm in Chen et al., then we prove Theorem 4.3. The proof of Theorem 4.2 is almost identical, and is deferred to Appendix A.2.

Single-Commodity Confluent Flow and The Algorithm of Chen et al. 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
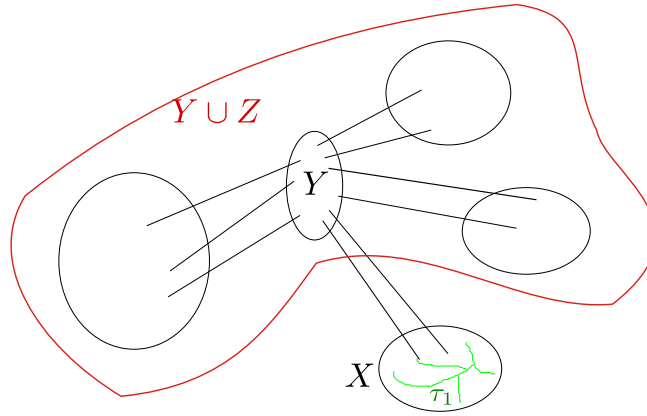
Fig. 3. The scenario in Algorithm 1 when the graph has low connectivity. The vertex set Y is a global minimum vertex cut of the graph. The vertex set X is the smallest connected component after the removal of Y, and Z is the union of all the other connected components.
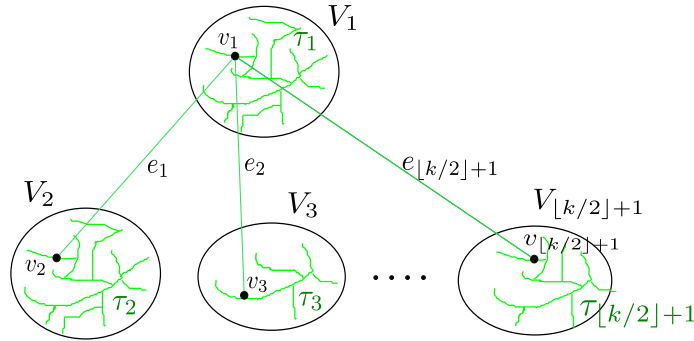
Fig. 4. The scenario in Algorithm 1 when the graph has high connectivity.

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. (Here, 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.

Theorem 4.4 ([14, 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 4.5. *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 $4(1 + \ln \ell)m/\ell$.*

Corollary 4.5 follows from Theorem 4.4 and Proposition 1.4. See Appendix A.1 for details.

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 4.5 and Lemma 4.1, `FindLCST` returns a tree with congestion at most $16m_H\sqrt{n_H/\hat{m}} \cdot \log n_H \leq 16\sqrt{\hat{m}n_H} \cdot \log n_H$.

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

$$\max\{ \text{ congestion of } \tau_1 \text{ in } H[X] \text{ , congestion of } \tau_2 \text{ in } H[Y \cup Z] \} + |X| \cdot |Y|$$

$$\leq 16\sqrt{\hat{m}(n_H - x)}\log(n_H - x) + \left(\sqrt{\hat{m}/n_H} + 1\right) \cdot x. \tag{1}$$

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

$$16\sqrt{\hat{m}(n_H - 1)}\log(n_H - 1) + \sqrt{\hat{m}/n_H} + 1 \leq 16\sqrt{\hat{m}n_H}\log n_H, \text{ as desired by Theorem 4.3.}$$

Runtime Analysis. At every internal node of the recursion tree, the algorithm makes two recursive calls with two vertex-disjoint and strictly smaller (w.r.t. vertex size) inputs. The dominating knitting cost is in Line 5 for computing a global minimum vertex cut, which is well-known that it can be done in polynomial time. Since at every leaf of the recursion tree the running time is polynomial, by standard analysis on divide-and-conquer algorithms, the running time of the whole algorithm is polynomial, which completes the proof of Theorem 4.3.

5   Lower Bound for Spanning Tree Congestion

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

Theorem 5.1. *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 $\Omega\left(\sqrt{mn}\right)$.*

We start with the following lemma, which states that for a random graph $\mathcal{G}(n, p)$, when p is sufficiently large, its *edge expansion* is $\Theta(np)$ with high probability. The proof of the lemma uses only fairly standard arguments and is deferred to Appendix A.3.

Lemma 5.2. *For any integer $n \geq 4$ and $1 \geq p \geq 32 \cdot \frac{\log n}{n}$, let $\mathcal{G}(n, p)$ denote the random graph with n vertices, in which each edge occurs independently with probability p. Then with probability at least $1 - \mathcal{O}(1/n)$, (i) the random graph is connected, (ii) the number of edges in the random graph is between $pn^2/4$ and pn^2, and (iii) for each subset of vertices S with $|S| \leq n/2$, the number of edges leaving S is at least $\frac{p}{2} \cdot |S| \cdot (n - |S|)$.*

In particular, for any sufficiently large integer n, when $n^2/2 \geq m \geq 16n \log n$, by setting $p = 2m/n^2$, there exists a connected graph 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 at least $\frac{m}{2n} \cdot |S| = \Theta(m/n) \cdot |S|$. We denote such a graph by $H(n, m)$.

We discuss our construction here (see Figure 5) before delving into the proof. 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 $H(n, m)$. The edge sets are denoted E_1, E_2, E_3 respectively. Up to this point, the construction is similar to that of Ostrovskii [30], 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 $(V_1 \cup V_2) \setminus \{v\}$. The set of these edges are denoted F_1. Similarly, for each vertex $v \in V_3 \cap V_2$, add an edge between v and every vertex in $(V_3 \cup V_2) \setminus \{v\}$. The set of these edges are denoted F_3. This new component is crucial: without it, we could only prove a lower bound of $\Omega(m/\sqrt{n}) = \Omega(\sqrt{mn} \cdot \frac{\sqrt{m}}{n})$.

Proof of Theorem 5.1: . Let $G = (V, E)$ be the graph constructed as above. The whole graph has $3n - 2\sqrt{m/n}$ vertices. The number of edges is at least m (due to edges in E_1 and E_3), and is at most $6m + 2\sqrt{m/n} \cdot 2n = 6m + 4\sqrt{mn}$, which is at most $7m$ for all sufficiently large n.

It is well known that for any tree on n vertices, there exists a vertex x called a centroid of the tree such that, removing x decomposes the tree into connected components, each of size at most $n/2$. Now, consider any spanning tree of the
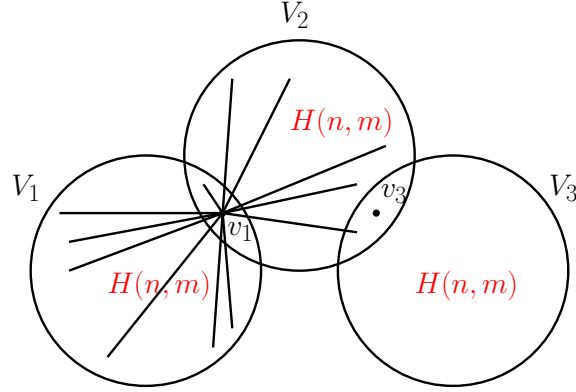
Fig. 5. Our lower-bound construction for spanning tree congestion. V_1, V_2, V_3 are three vertex subsets of the same size. In each of the subsets, we embed expander $H(n, m)$. There is a small overlap between V_2 and V_1, V_3, while V_1, V_3 are disjoint. For any vertex $v_1 \in V_1 \cap V_2$, we add edges between it and any other vertex in $V_1 \cup V_2$; similarly, for any vertex $v_3 \in V_3 \cap V_2$, we add edges (not shown in figure) between it and any other vertex in $V_3 \cup V_2$.

given graph, let u be a centroid of the tree. Without loss of generality, we can assume that $u \notin V_1$; otherwise we swap the roles of V_1 and V_3. The removal of u (and its adjacent edges) from the tree decomposes the tree into a number of connected components. For any of these components which intersects V_1, it must contain at least one vertex of $V_1 \cap V_2$, thus the number of such components is at most $\sqrt{m/n}$, and hence there exists one of them, denoted by U_j, such that

$$b_1 := |U_j \cap V_1| \geq n/(\sqrt{m/n}) = n\sqrt{n/m}.$$

Let e_j denote the tree-edge that connects u to U_j. Then there are three cases:

Case 1: $n\sqrt{n/m} \leq b_1 \leq n - n\sqrt{n/m}$. Due to the property of $H(n, m)$, the congestion of e_j is at least $\Theta(m/n) \cdot \min\{b_1, n - b_1\} \geq \Theta(\sqrt{mn})$.

Case 2: $b_1 > n - n\sqrt{n/m}$ and $|U_j \cap V_1 \cap V_2| \leq \frac{1}{2} \cdot \sqrt{m/n}$. Let $W := (V_1 \cap V_2) \setminus U_j$. Note that by this case's assumption, $|W_1| \geq \frac{1}{2} \cdot \sqrt{m/n}$. Due to the edge subset F_1, the congestion of e_j is at least

$$\left| F_1(W, V_1 \setminus W) \right| \geq \left(\frac{1}{2} \cdot \sqrt{m/n} \right) \cdot \frac{n}{2} = \Theta\left(\sqrt{mn} \right).$$

Case 3: $b_1 > n - n\sqrt{n/m}$ and $|U_j \cap V_1 \cap V_2| > \frac{1}{2} \cdot \sqrt{m/n}$. Let $W' := U_j \cap V_1 \cap V_2$, and let $Z := (V_2 \setminus V_1) \cap U_j$.

Note that $b_1 > n - n\sqrt{n/m} \geq 9n/10$. Suppose $|Z| \geq 6n/10$, then $|U_j| > 9n/10 + 6n/10 > |V|/2$, a contradiction to the assumption that u is a

centroid. Thus, $|Z| < 6n/10$. Due to the edge subset F_2, the congestion of e_j is at least

$$
\left| F_2(W' \cup Z \ , \ V_2 \setminus (W' \cup Z)) \right| \ \geq \ |W'| \cdot (n - |W'| - |Z|)
$$

$$
\geq \ \left(\frac{1}{2} \cdot \sqrt{m/n} \right) \cdot \left(n - \sqrt{m/n} - \frac{6n}{10} \right) \ = \ \Theta(\sqrt{mn}).
$$

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 6.1. *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 6.2. *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*

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

Next, we present the polynomial time algorithm in Theorem 6.2 and its analysis.

Algorithm. Let G be an (n, s, d_1, d_2, d_3, t)-expanding graph. By Condition (2), every vertex has degree at least d_2. Let v_0 be a vertex of degree $d \geq d_2$, and let $v_1, \cdots, v_d$ be its d neighbours. We maintain a tree T rooted at v_0 such that $T = T_1 \cup T_2 \cup \cdots \cup T_d \cup \{v_0 v_1, v_0 v_2, \ldots, v_0 v_d\}$ where $T_1, T_2, \cdots, T_d$ are trees rooted at $v_1, v_2, \ldots, v_d$ respectively. We call the $T_i's$ as <u>branches</u>. (See Figure 6). We start with each branch $T_i = v_i$. In order to minimize congestion, we grow T in a balanced way, i.e., we maintain that the T_i's are roughly of the same size. A branch is <u>saturated</u> if it contains at least $\max \left\{ s + 1 \ , \ \frac{3d_1 n}{d_2} \right\}$ vertices.

At any point of time, let V_T be the set of vertices in T and $\overline{V_T}$ be the vertices not in T. Often, we will move a subtree of a saturated branch T_i to an unsaturated branch T_j to ensure balance. For any $x \in V_T$, let T_x denote the subtree of T rooted at x. A vertex x of a saturated branch T_i is called <u>transferable</u> (to branch T_j) if x has a neighbour y in T_j and the tree $T_j \cup \{xy\} \cup T_x$ is unsaturated. (See Figure 7.)
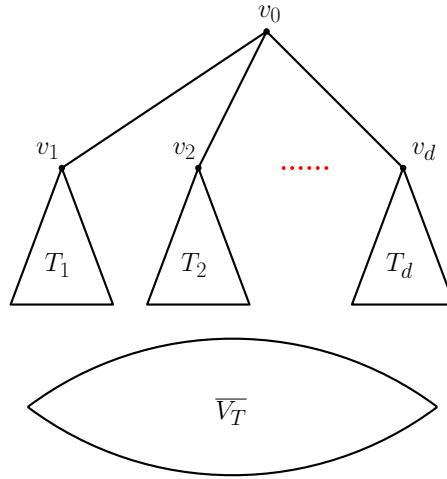
Fig. 6. The tree T and its branches
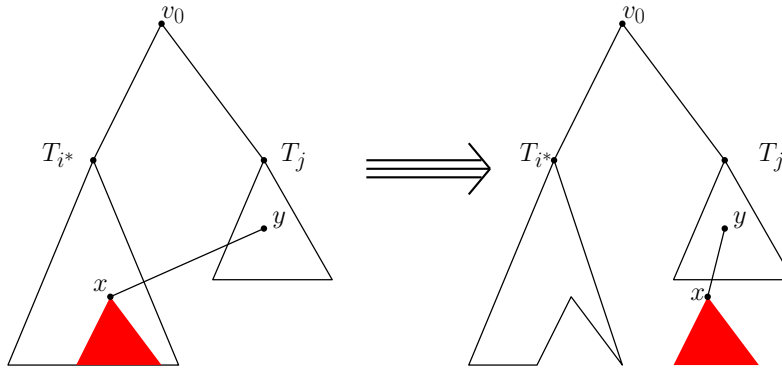
Fig. 7. Transfer of a subtree from a saturated branch to an unsaturated branch

The algorithm is divided into two phases which are described below. Throughout the algorithm, whenever a branch T_i gets modified, T gets modified accordingly, and whenever T gets modified V_T and $\overline{V_T}$ gets modified accordingly.

Phase 1: Repeatedly do one of the following two actions, until $|V_T| \geq d_1 n$: (We will prove that the precondition of at least one of the actions is satisfied if $|V_T| < d_1 n$)

1. If there exists a $b \in \overline{V_T}$ such that b has a neighbour a in some unsaturated branch T_i:
   Add the vertex b and the edge ab to branch T_i.
2. If there exists at least one transferable vertex: (see Figure 7)
   Find the transferable vertex x such that T_x is the smallest. Let T_{i^*} be the

branch currently containing x, T_j be a branch to which it is transferable, and y be an arbitrarily chosen neighbour of x in T_j.

(a) Remove the subtree T_x from T_{i^*} and add it to T_j with x as a child of y.
(b) Pick a $b \in \overline{V_T}$ that has a neighbour a (arbitrarily chosen, if many) either in T_{i^*} or in T_j. (We will show in the analysis that such b exists). We add vertex b and edge ab to the branch containing a (i.e. to T_{i^*} or T_j).

Phase 2: While $\overline{V_T} \neq \emptyset$, repeat:
Find a maximum matching of $G[V_T, \overline{V_T}]$, the bipartite graph formed by edges of G between V_T and $\overline{V_T}$. Let M be the matching. Add all edges of M to T.

In the analysis below, we say that a tree is <u>saturated</u> if it contains at least $\mathcal{A}$ vertices; we will determine its appropriate value by the end of the analysis.

Analysis of Phase 1. We claim that during Phase 1, i.e. if $|V_T| < d_1 n$, the precondition of either step 1 or step 2 is satisfied. We also show the existence of a vertex b as specified in step 2b, whenever step 2b is reached. Given these and the fact that a vertex in $\overline{V_T}$ is moved to V_T (either in step 1 or in step 2b) during each round of Phase 1, we have that Phase 1 runs correctly and terminates after a linear number of rounds.

During Phase 1, we will also maintain the invariant that each branch has at most $\mathcal{A}$ vertices; thus, each saturated branch has exactly $\mathcal{A}$ vertices. We call this invariant the <u>balancedness</u>. Note that balancedness is not violated due to step 1, as the new vertex is added to an unsaturated branch. It is not violated during step 2 as the branches T_{i^*} and T_j (as defined in step 2) become unsaturated at the end of the step.

We define the <u>hidden vertices</u> of T (denoted by $H \equiv H_T$) as follows: they are the vertices which are not adjacent to any vertices outside the tree, i.e., to any vertex in $\overline{V_T}$. If there is an unsaturated branch with a non-hidden vertex, clearly the precondition of step 1 is satisfied. So, let us assume that all the vertices in all unsaturated branches are hidden. In such a case, we show that the precondition of step 2 is satisfied if $|V_T| < d_1 n$.

We argue that in this case $|H| \leq s$: otherwise, take a subset $H' \subset H$ of cardinality s, then by condition (2), $N(H')$, which is contained in V_T, has cardinality at least $d_1 n$, a contradiction.

Since $|V_T| < d_1 n$, the number of saturated branches is at most $d_1 n / \mathcal{A}$. To ensure that at least one unsaturated branch exists, we set $\mathcal{A}$ such that $d_1 n / \mathcal{A} < d_2$. Let U denote the set of vertices in all unsaturated branches. Since all vertices in U are hidden vertices, $|U| \leq s$. Then by condition (2), $|N(U)| \geq d_2 |U|$. Note that the vertices in $N(U)$ are all in the saturated branches. By the pigeon-hole principle, there exists a saturated branch containing at least

$$N(U)/(d_1 n / \mathcal{A}) \ \geq \ \frac{\mathcal{A} d_2 |U|}{d_1 n}$$

vertices of $N(U)$. By setting $\mathcal{A} \geq \frac{3 d_1 n}{d_2}$, the above calculation guarantees the existence of a saturated branch containing at least $3|U| \geq |U| + 2$ vertices of $N(U)$; let T_i be such a branch.

In T_i, pick a vertex $x \in T_i \cap N(U)$ such that T_x does not contain any vertex in $N(U)$, except x. Then the size of T_x is at most $\mathcal{A} - |N(U) \cap T_i| + 1 \geq \mathcal{A} - (|U| + 1)$. Let $y \in U$ be a vertex which is adjacent to x and T_j be the branch containing y. Since T_j has at most $|U|$ vertices, x is a transferable vertex (to T_j). Thus precondition of step 2 is satisfied.

We further set $\mathcal{A} > s$ so that in each saturated branch, there is at least one unhidden vertex. In particular, T_i has an unhidden vertex, which is adjacent to some $b \in \overline{V_T}$. The vertex b is either adjacent to a vertex in T_x, or a vertex in $T_i \setminus T_x$ as required in step 2b.

Analysis of Phase 2. Since G is connected, M is non-empty in each iteration of Phase 2, and hence Phase 2 terminates in linear number of rounds. At the end of Phase 2, since $\overline{V_T}$ is empty, T is clearly a spanning tree. It only remains to estimate the congestion of this spanning tree. Towards this, we state the following *modified Hall's theorem*, which is an easy corollary of the standard Hall's theorem.

Lemma 6.3. *In a bipartite graph (L, R) with $|L| \leq |R|$, for any vertex $w \in L$, let $R(w)$ denote the neighbours of w in R; then for any $W \subset L$, let $R(W) := \cup_{w \in W} R(w)$. Suppose that there exist $t \geq 0$ such that for any $W \subset L$, we have $|R(W)| \geq |W| - t$. Then the bipartite graph admits a matching of size at least $|L| - t$.*

Recall that Phase 2 consists of multiple rounds of finding a matching between V_T and $\overline{V_T}$. As long as $|V_T| \leq n/2$, condition (3) (with $S = V_T$) plus the modified Hall's theorem (with $L = V_T$ and $R = \overline{V_T}$) guarantees that in each round, at least

$$|V_T| - t \ \geq \ \left(1 - \frac{t}{d_1 n}\right) \cdot |V_T| \ =: \ (1 - \delta)|V_T|$$

number of vertices in $\overline{V_T}$ are matched. Thus, after at most $\left\lceil \log_{(2-\delta)} \frac{1}{2d_1} \right\rceil$ rounds of matching, $|V_T| \geq n/2$. After reaching $|V_T| \geq n/2$, condition (3) (with $S = \overline{V_T}$) plus the modified Hall's theorem (with $L = \overline{V_T}$ and $R = V_T$) guarantees that after one more round of matching, all but t vertices are left in $\overline{V_T}$.

By the end of Phase 1, each branch had at most $\mathcal{A}$ vertices. After each round of matching, the cardinality of each branch is doubled at most. Thus, the maximum possible number of vertices in each branch after running the whole algorithm is at most

$$\mathcal{A} \cdot 2^{\left\lceil \log_{(2-\delta)} \frac{1}{2d_1} \right\rceil + 1} + t \ \leq \ 4\mathcal{A} \cdot \left(\frac{1}{2d_1}\right)^{\log_{(2-\delta)} 2} + t.$$

and hence the STC is at most

$$d_3 \cdot \left[4\mathcal{A} \cdot \left(\frac{1}{2d_1}\right)^{\log_{(2-\delta)} 2} + t \right].$$

Recall that we need $\mathcal{A}$ to satisfy $d_1 n / \mathcal{A} < d_2$, $\mathcal{A} \geq \frac{3d_1 n}{d_2}$ and $\mathcal{A} > s$. Thus we set $\mathcal{A} := \max\left\{ s + 1 \ , \ \left\lceil \frac{3d_1 n}{d_2} \right\rceil \right\}$.

6.1   Random Graph

Let $G \in \mathcal{G}(n, p)$ where $p \geq c_0 \log n / n$, and $c_0 = 64$. The following lemmas show that with high probability G 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 $\delta = o(1)$). The proof of the lemmas are deferred to Appendix B.2.

Lemma 6.4. *For any $S \subseteq V(G)$ such that $|S| = \lceil 1/p \rceil$, we have $|N(S)| \geq c_2 n$ with probability at least $1 - e^{-n/16}$, where $c_2 = 1/25$.*

Lemma 6.5. *For any $S \subseteq V(G)$ such that $|S| \leq 1/p$, we have $|N(S)| \geq c_3 np|S|$ with probability at least $1 - \mathcal{O}(1/n^2)$, where $c_3 = 1/16$.*

Lemma 6.6. *For all $A \subseteq V(G)$ such that $|A| \leq n/2$, and for all $S \subseteq A$, with probability at least $1 - e^{-n}$, S has at least $|S| - c_4/p$ neighbors in $V(G) \setminus A$, where $c_4 = 12$.*

Lemma 6.7. *For all $S \subseteq V(G)$, the cut size $|E(S, V(G) \setminus S)|$ is at most $np|S|$ with probability at least $1 - n^{-c_0/4}$.*

Plugging the bounds from above lemmas into Theorem 6.2, together with a separate lower bound argument (Theorem B.2 in Appendix B.1), we have the following theorem; in Appendix B.1, we also present a non-algorithmic proof of this theorem.

Theorem 6.8. *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 $\Theta(n)$.*

7   Discussion and Open Problems

In this paper, we provide thorough understanding, both combinatorially and algorithmically, on the spanning tree congestion of general graphs and random graphs. On course of doing so, we also provide the first constructive proof for the generalized Győri-Lovász theorem, which might be of independent interest. Following are some natural open problems:

– Finding the spanning tree with minimum congestion is NP-hard; indeed, Bodlaender et al. [9] showed a $(9/8 - \epsilon)$-approximation NP-hardness for the STC problem. Does a constant or a poly-logarithmic factor approximation polynomial time algorithm exist?
– We present an algorithm for computing a spanning tree achieving congestion at most $\mathcal{O}(\sqrt{mn})$. The algorithm runs in sub-exponential time when $m = \omega(n \log^2 n)$. Is there a polynomial time algorithm for constructing such a spanning tree?
– For a k-connected graph, a connected k-partition where all parts are of size at most $\mathcal{O}((n/k) \log k)$ can be found in polynomial time due to an algorithm of Chen et al. [14]. Can we improve the sizes of parts to $\mathcal{O}(n/k)$?
– Is finding Győri-Lovász partition PLS-complete? If not, is it polynomial time solvable?

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. Ingo Althöfer, Gautam Das, David P. Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993.
5. Reid Andersen and Uriel Feige. Interchanging distance and capacity in probabilistic mappings. *CoRR*, abs/0907.3631, 2009.
6. Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. Expander flows, geometric embeddings and graph partitioning. *J. ACM*, 56(2):5:1–5:37, 2009.
7. 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.
8. 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.
9. 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.
10. Hans L. Bodlaender, Kyohei Kozawa, Takayoshi Matsushima, and Yota Otachi. Spanning tree congestion of k-outerplanar graphs. *Discrete Mathematics*, 311(12):1040–1045, 2011.
11. Béla Bollobás. *Random Graphs*. Cambridge University Press, 2001.
12. Béla Bollobás and Andrew Thomason. Random graphs of small order. *North-Holland Mathematics Studies*, 118:47–97, 1985.
13. Leizhen Cai and Derek G. Corneil. Tree spanners. *SIAM J. Discrete Math.*, 8(3):359–387, 1995.
14. 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.
15. Michael Elkin, Yuval Emek, Daniel A. Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. *SIAM J. Comput.*, 38(2):608–628, 2008.
16. Uriel Feige and Robert Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM J. Comput.*, 31(4):1090–1118, 2002.
17. E. Győri. On division of graphs to connected subgraphs. *Colloq. Math. Soc. Janos Bolyai*, 18:485–494, 1976.
18. Ludovic Hofer and Thibaud Lambert. Study of the article: An $O(k^2 n^2)$ algorithm to find a k-partition in a k-connected graph. 2014.
19. Alexander Hoyer and Robin Thomas. The Győri-Lovász theorem. *arXiv*, abs/1605.01474, 2016. URL: `http://arxiv.org/abs/1706.08115`.
20. David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988.
21. Marcos A. Kiwi, Daniel A. Spielman, and Shang-Hua Teng. Min-max-boundary domain decomposition. *Theor. Comput. Sci.*, 261(2):253–266, 2001.

22. 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.
23. Kyohei Kozawa and Yota Otachi. Spanning tree congestion of rook's graphs. *Discussiones Mathematicae Graph Theory*, 31(4):753–761, 2011.
24. Kyohei Kozawa, Yota Otachi, and Koichi Yamazaki. On spanning tree congestion of graphs. *Discrete Mathematics*, 309(13):4215–4224, 2009.
25. Hiu Fai Law, Siu Lam Leung, and Mikhail I. Ostrovskii. Spanning tree congestions of planar graphs. *Involve*, 7(2):205–226, 2014.
26. László Lovász. A homology theory for spanning trees of a graph. *Acta Math. Acad. Sci. Hungaricae*, 30(3–4):241–251, 1977.
27. Christian Löwenstein, Dieter Rautenbach, and Friedrich Regen. On spanning tree congestion. *Discrete Math.*, 309(13):4653–4655, 2009.
28. Shin-Ichi Nakano, Md. Saidur Rahman, and Takao Nishizeki. A linear-time algorithm for four-partitioning four-connected planar graphs. *Inf. Process. Lett.*, 62(6):315–322, 1997.
29. 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.
30. M. I. Ostrovskii. Minimal congestion trees. *Discrete Math.*, 285:219–226, 2004.
31. M. I. Ostrovskii. Minimum congestion spanning trees in planar graphs. *Discrete Math.*, 310:1204–1209, 2010.
32. M. I. Ostrovskii. Minimum congestion spanning trees in bipartite and random graphs. *Acta Mathematica Scientia*, 31(2):634 – 640, 2011.
33. Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *STOC*, pages 255–264, 2008.
34. André Raspaud, Ondrej Sýkora, and Imrich Vrto. Congestion and dilation, similarities and differences: A survey. In *SIROCCO 2000*, pages 269–280, 2000.
35. Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.
36. Shai Simonson. A variation on the min cut linear arrangement problem. *Mathematical Systems Theory*, 20(4):235–252, 1987.
37. Daniel A. Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM J. Comput.*, 42(1):1–26, 2013.
38. David Steurer. Tight bounds for the min-max boundary decomposition cost of weighted graphs. In *SPAA 2006*, pages 197–206, 2006.
39. Hitoshi Suzuki, Naomi Takahashi, and Takao Nishizeki. A linear algorithm for bipartition of biconnected graphs. *Inf. Process. Lett.*, 33(5):227–231, 1990.
40. Zoya Svitkina and Éva Tardos. Min-max multiway cut. In *APPROX-RANDOM 2004*, pages 207–218, 2004.
41. Shang-Hua Teng. Scalable algorithms for data and network analysis. *Foundations and Trends in Theoretical Computer Science*, 12(1-2):1–274, 2016.
42. 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.

A   Missing Proofs in Sections 4 and 5

A.1   Proof of Corollary 4.5

First of all, we set the demand of each vertex in the flow problem to be the the degree of the vertex in G, and $t_1, \cdots, t_\ell$ as the sinks in the flow problem.

By Proposition 1.4, there exists an ℓ-connected-partition $\cup_{j=1}^{\ell} U_\ell$ such that for all $j \in [\ell]$, $t_j \in U_j$, and the total degrees of vertices in each U_j is at most $4m/\ell$. With this, by routing the demand of a vertex in U_j to t_j via an arbitrary path in $G[U_j]$ only, we construct a splittable flow with node congestion at most $4m/\ell$. By Theorem 4.4, one can construct a confluent flow with node congestion at most $4(1 + \ln \ell)m/\ell$ in polynomial time.

Obviously, in the confluent flow, all the flow originating from one vertex goes completely into one sink. Set V_j to be the set of vertices such that the flows originating from these vertices go into t_j. It is then routine to check that $\cup_{j=1}^{\ell} V_\ell$ is our desired ℓ-connected-partition.

A.2   Proof of Theorem 4.2

Instead of giving the full proof, we point out the differences from the proof of Theorem 4.3.

First, in handling the basis case (ii), by Theorem 1.3(b), Proposition 1.4 and Lemma 4.1, we have an improved upper bound on the congestion of the returned tree, which is $8m_H/\sqrt{\hat{m}/n_H} \leq 8\sqrt{\hat{m}n_H}$. Thus, (1) can be improved to

$$8\sqrt{\hat{m}(n_H - x)} \; + \; \sqrt{\frac{\hat{m}}{n_H}} \cdot x.$$

Again, by viewing x as a real variable and taking derivative, it is easy to see that the above expression is maximized at $x = 1$. So the above bound is at most

$$8\sqrt{\hat{m}(n_H - 1)} \; + \; \sqrt{\frac{\hat{m}}{n_H}} \;\; \leq \;\; 8\sqrt{\hat{m}n_H}, \;\; \text{as desired.}$$

Concerning the running time, it is clear that in the worst case, it is dominated by some calls to the algorithm in Theorem 1.3(b). Note that the number of such calls is at most $\hat{n}$, since each call to the algorithm is on a disjoint set of vertices.

There remains one concern, which is the connectedness of $H[Y \cup Z]$. Suppose the contrary that $H[Y \cup Z]$ is not connected. Let C be one of its connected components, so that it contains the least number of vertices from Y. Then C contains at most $\lfloor |Y|/2 \rfloor$ vertices from Y, i.e., $|C \cap Y| < |Y|$. Note that $C \cap Y$ is a vertex cut set of the graph H, thus contradicting that Y is a global minimum vertex cut set.

A.3   Proof of Lemma 5.2

It is well known that the requirements (i) and (ii) are satisfied with probability $1 - o(1/n)$. [11] For each subset S with $|S| \leq n/2$, by the Chernoff bound,

$$\mathbb{P}\left[\ \left|E(S, V \setminus S)\right| \ \leq \ \frac{p}{2} \cdot |S| \cdot (n - |S|)\ \right] \ \leq \ e^{-p|S|(n-|S|)/8} \ \leq \ e^{-pn|S|/16}.$$

Since $p \geq 32 \cdot \frac{\log n}{n}$, the above probability is at most $n^{-2|S|}$. Then by a union bound, the probability that (iii) is not satisfied is at most

$$\sum_{s=1}^{\lfloor n/2 \rfloor} \binom{n}{s} \cdot n^{-2s} \ \leq \ \sum_{s=1}^{\lfloor n/2 \rfloor} n^s \cdot n^{-2s} \ \leq \ \sum_{s=1}^{\lfloor n/2 \rfloor} n^{-s} \ \leq \ \frac{2}{n}.$$

B   Spanning Tree Congestion of Random Graphs

B.1   Non-Algorithmic Proof of Theorem 6.8

We first present a simple non-algorithmic proof that random graph has STC $\Theta(n)$ with high probability. Theorem B.1 gives the upper bound and Theorem B.2 gives the lower bound. The proof of Theorem B.1 uses Lemma 1.5 and the fact that for random graphs, vertex-connectivity and minimum degree are equal with high probability. Theorem B.1 does not give an efficient algorithm.

Theorem B.1. *If $G \in \mathcal{G}(n, p)$ where $p \geq 8 \log n/n$, then the spanning tree congestion of G is at most $16n$ with probability at least $1 - o(1/n)$.*

Proof. It is known that the threshold probability for a random graph being k-connected is same as the threshold probability for it having minimum degree at least k [12]. Since $p \geq 8 \log n/n$, using Chernoff bound and taking union bound over all vertices gives that G has minimum degree at least $np/2$ with probability at least $1 - o(1/n)$. Hence G is $(np/2)$-connected with probability at least $1 - o(1/n)$. We also have that the number of edges in G is at most $2n^2p$ with probability at least $1 - o(1/n)$. Now, by using Lemma 1.5, we have that with probability at least $1 - o(1/n)$, the spanning tree congestion is at most $16n$.   □

Theorem B.2. *If $G \in \mathcal{G}(n, p)$ where $p \geq 32 \log n/n$, then the spanning tree congestion of G is $\Omega(n)$ with probability $1 - \mathcal{O}(1/n)$.*

Proof. By using Chernoff Bounds and applying union bound, it is easy to show that with probability $1 - o(1/n)$, every vertex of G has degree at most $c_1 np$ for a sufficiently large constant c_1. Also, by Lemma 5.2, with probability $1 - \mathcal{O}(1/n)$, properties (i) and (iii) of that lemma holds. In the proof below, we conditioned on the above mentioned highly probable events.

Take a spanning tree T of G which gives the minimum congestion. Let u be a centroid of the tree T, i.e., each connected component of $T \setminus \{u\}$ has at most $n/2$ vertices. If there is a connected component with number of vertices at least

$n/4$, then define this connected component as T'. Else, all connected components have at most $n/4$ vertices. In this case, let T' be the forest formed by the union of a minimum number of connected components of $T \setminus \{u\}$ such that $|T'| \geq n/4$. It is easy to see that $|T'| \leq n/2$. Also, the number of edges in T from $V(T')$ to $V(T) \setminus V(T')$ is at most $\deg_G(u)$, which is at most $c_1 np$.

By property (iii) of Lemma 5.2, the number of edges between $V(T')$ and $V(G) \setminus V(T')$ is $\Omega(n^2 p)$. Each of these edges in G between $V(T')$ and $V(G) \setminus V(T')$ have to contribute to the congestion of at least one of the edges in T between $V(T')$ and $V(G) \setminus V(T')$. Now since T' sends at most $c_1 np$ tree edges to other parts of T, it follows that there exists one edge in T with congestion at least $\Omega(n^2 p)/(c_1 np) = \Omega(n)$, as claimed. □

B.2   Random Graph Satisfies Expanding Properties

Constants. For easy reference, we list out the constants used.

$$c_0 = 64, \ c_2 = 1/25, \ c_3 = 1/16, \ c_4 = 12$$

Proof of Lemma 6.4: . Let $\overline{S} = V(G) \setminus S$. The probability that a fixed vertex in $\overline{S}$ does not have edge to S is at most $(1-p)^{|S|} \leq (1-p)^{1/p} \leq e^{-1}$. Since $|\overline{S}| \geq n - 2/p \geq n - 2n/(c_0 \log n) \geq 31n/32$, the expected value of $|N(S)|$ is at least $(31/32) n(1 - e^{-1}) \geq n/2$. Hence, using Chernoff bound, the probability that $|N(S)| < c_2 n = n/25$ is at most $e^{-n/8}$. Since the number of such S is at most $n^{2/p} = 2^{2n/c_0} \leq 2^{n/32}$, we have the lemma by applying union bound.   □

Proof of Lemma 6.5: . Let $\overline{S} = V(G) \setminus S$. Since $|S| \leq 1/p \leq n/\log n$, we have $|\overline{S}| \geq n/2$ for sufficiently large n. Divide $\overline{S}$ into groups of size $\lceil 1/(p|S|) \rceil$. The probability that such a group does not have edge to S is at most $(1-p)^{|S|(1/(p|S|))} \leq 1/e$. The expected number of groups having edge to S is at least $(np|S|/2)(1 - 1/e) \geq np|S|/4$. Thus, by Chernoff bound, the probability that $|N(S)| \leq np|S|/16$ is at most $e^{-np|S|/16} \leq 2^{-c_0|S|\log n/16} \leq 2^{-4|S|\log n}$. The number of sets of size $|S|$ is at most $2^{|S|\log n}$. Hence, taking union bound over all S with $|S| \leq 1/p$, we get the required lemma.   □

Proof of Lemma 6.6: . First, we prove that for all $C, D \subseteq V(G)$ such that $|C| \geq n/4, |D| \geq c_4/p$, and $C \cap D = \emptyset$, there exist at least one edge between C and D with high probability. The probability that there is no edge between such a fixed C and D is at most $(1-p)^{(n/4)(c_4/p)} \leq e^{-c_4 n/4}$. The number of pairs of such C and D is at most 2^{2n}. Hence, by taking union bound, the probability that for all C and D, the claim holds is at least $1 - e^{2n-(c_4 n/4)} \geq 1 - e^{-n}$.

Using the above claim, we prove that for all $S \subseteq A$, S has at least $|S| - c_4/p$ neighbors in $\overline{A} := V(G) \setminus A$ with high probability. Suppose there is an S which violates the claim. Note that we can assume $|S| \geq c_4/p$, because otherwise the claim is vacuously true. Let $B := \overline{A} \setminus N(S)$. There cannot be any edges between S and B. Also, $|B| \geq (n/2) - (|S| - (c_4/p))$. So, $|B|$ is at least c_4/p and when $|B| < n/4$, $|S|$ is at least $n/4$. Hence, using the previous claim, there is an edge between S and B with probability at least $1 - e^{-n}$. Hence, we get a contradiction, and hence our claim is true with probability at least $1 - e^{-n}$.   □

Proof of Lemma 6.7: . Let $\mathcal{C}(S)$ denote $|E(S, V(G) \setminus S)|$. For a fixed vertex subset S, the expected value of $\mathcal{C}(S)$ is at most $np|S|$. Therefore, probability that $\mathcal{C}(S) > np|S| \geq c_0|S|\log n$ is at most $n^{-c_0|S|/2}$ using Chernoff bounds. The probability that $\mathcal{C}(S) \leq np|S|$ for all sets S of size k is at least $1 - n^{-c_0 k/2+k} \geq 1 - n^{-c_0/2+1}$ using union bound and using $k \geq 1$. The probability that $\mathcal{C}(S) \leq np|S|$ for all vertex subsets S is at least $1 - n^{-c_0/2+2} \geq$ using union bound over all $k \in [n]$. □