Two Complexity Results on Spanning-Tree Congestion Problems*

Sunny Atalig†, Marek Chrobak†, Christoph Dürr◇, Petr Kolman§, Huong Luu‡, Jǐrí Sgall§, and Gregory Zhu†

†University of California at Riverside, USA
‡California Polytechnic University at Pomona, USA
§Charles University, Prague, Czech Republic
◇Sorbonne Université, CNRS, LIP6, France

January 15, 2026

Abstract

In the spanning-tree congestion problem (STC), given a graph G, the objective is to compute a spanning tree of G for which the maximum edge congestion is minimized. While STC is known to be $\mathbb{NP}$-hard, even for some restricted graph classes, several key questions regarding its computational complexity remain open, and we address some of these in our paper. (i) For graphs of maximum degree Δ, it is known that STC is $\mathbb{NP}$-hard when $\Delta \geq 8$. We provide a complete resolution of this variant, by showing that STC remains $\mathbb{NP}$-hard for each degree bound $\Delta \geq 3$. (ii) In the decision version of STC, given an integer K, the goal is to determine whether the congestion of G is at most K. We prove that this variant is polynomial-time solvable for K-edge-connected graphs.

1 Introduction

Constructing spanning trees for graphs under specific constraints is a well-studied problem in graph theory and algorithmics. In this paper, we focus on *the spanning-tree congestion problem (STC)*, that arises naturally in some network design and routing problems. The problem can be viewed as a special case of the graph sparsification problem where a graph G is embedded into its spanning tree T by mapping each edge (x, y) of G to the unique x-to-y path in T. The congestion of an edge $e \in T$ is defined as the number of edges of G whose corresponding path in T traverses e, and the congestion of T is the maximum congestion of its edges. In the STC problem, we are given a graph G and the objective is to compute a spanning tree with minimum congestion. This minimum congestion value is referred to as the *spanning-tree congestion of* G and denoted by $\mathrm{stc}(G)$.

The concept of spanning-tree congestion was introduced under different names in the late 1990s [1, 30, 28, 13], and in 2003 formalized by Ostrovskii [25], who established some key properties. The problem has been extensively studied since then, and numerous results regarding its graph-theoretic properties and computational complexity have been reported in the literature. Below we review briefly those that are most relevant to our work.

*Research supported by NSF grant CCF-2153723 (M. Chrobak), ANR grant ANR-23-CE48-0010 (C. Dürr), and grant 24-10306S of GA ČR (J. Sgall and P. Kolman).

STC is $\mathbb{NP}$-hard, with the first $\mathbb{NP}$-hardness proof given by Löwenstein [22] in 2010. It remains $\mathbb{NP}$-hard for planar graphs [27], chain graphs and split graphs [24]. On the other hand, STC is polynomial-time solvable for a wide variety of special graph classes, including complete k-partite graphs, two-dimensional tori [17], outerplanar graphs [3], two-dimensional Hamming graphs [16], co-chain graphs [11], and interval graphs [21].

In the decision version of STC, in addition to a graph G we are also given an integer K, and the goal is to determine if $\mathrm{stc}(G) \leq K$. A natural variant of STC, when the congestion parameter K is a fixed constant (rather than given as input) is denoted K-STC. The K-STC problem was shown to be $\mathbb{NP}$-complete for $K \geq 5$ by Luu and Chrobak [23], building on earlier results for larger constants [27, 2]. On the other hand, K-STC is solvable in linear time for $K \leq 3$ [27]. The complexity status of 4-STC remains an intriguing open problem. For graphs of radius 2, K-STC is $\mathbb{NP}$-complete for $K \geq 6$ [23], and its complexity is open for $K = 4, 5$. For any constant K, K-STC is linear-time solvable for bounded-degree graphs, bounded-treewidth graphs, apex-minor-free graphs [2], and chordal graphs [26].

Kolman [14] observed that the existing $\mathbb{NP}$-hardness proofs used graphs of unbounded degree, and raised the question about the complexity of STC for graphs of constant degree. For constant-degree graphs, it has only been known that K-STC is linear-time solvable if the congestion bound K is also constant [2]. Recently, Lampis et al. [18] reported progress on this problem, by proving that, for any constant $\Delta \geq 8$, STC is $\mathbb{NP}$-hard for graphs with maximum degree Δ, leaving open the complexity of STC for degree bounds between 3 and 7.

Our contributions. Addressing the problem left open by Lampis et al. [18], we prove (see Section 3) the following theorem:

Theorem 1.1. *Problem STC is $\mathbb{NP}$-hard for graphs of maximum degree at most* 3*.*

Naturally, this theorem is true for all degree bounds $\Delta \geq 3$, and it remains true for 3-regular graphs. (Vertices of degree 1 or 2 can be removed from the graph, without affecting its maximum congestion value.) This result fully resolves the status of STC for bounded-degree graphs.

We also study the K-STC problem for K-edge-connected graphs (see Section 5), proving the following theorem:

Theorem 1.2. *There is an $\tilde{O}(m)$-time algorithm that, given a K-edge-connected graph G, determines whether* $\mathrm{stc}(G) = K$*.*

Above, m is the number of edges, and the $\tilde{O}(m)$ time bound is actually independent of K. Our solution is based on the so-called cactus representation of K-cuts in K-edge-connected graphs that was developed by Dinic et al. [6] (see also [9]). We further refine this characterization for graphs with congestion K, to obtain additional properties that lead to a fast algorithm. Besides its own interest, this result sheds new light on the complexity of 4-STC, showing that its difficulty is related to the presence of cuts of size less than 4 in the graph.

Other related work. General bounds for the spanning-tree congestion value have been well studied. For graphs with n vertices and m edges it is known that the congestion is at most $O(\sqrt{mn})$ and that there are graphs where this value is $\Omega(\sqrt{mn})$ [5]. Since the congestion of an n-clique is $n-1$, this implies that, somewhat counter-intuitively, the congestion value is not monotone: adding edges can actually decrease the congestion, and quite substantially so.

This non-monotonicity is particularly challenging in the context of approximations. Indeed, very little is known about the approximability of STC. While the upper bound of $n/2$ on the approximation ratio is trivial (achieved by *any* spanning tree [26]), the best known lower bound is only 1.2, implied directly by the $\mathbb{NP}$-completess of 5-STC [23]. In a recent work, Kolman [15] developed an algorithm with approximation ratio $\tilde{O}(\Delta)$, where Δ is the maximum vertex degree. Yet the general problem remains open; in particular it is not known if it is possible to achieve ratio $O(n^\delta)$, for some $\delta < 1$.

The spanning-tree congestion is related to the tree spanner problem which seeks a spanning tree with minimum stretch factor. The two problems are in fact equivalent in the case of planar graphs: the spanning-tree congestion of a planar graph is equal to the minimum stretch factor of its dual plus one [8, 27] (cf. [4, 8, 7] and the references therein for further discussion). It is worth mentioning here that the complexity status of the tree 3-spanner problem has remained open since its introduction in 2014 [4].

The STC problem can be relaxed by dropping the restriction that the tree to be computed is a spanning tree of the graph. In this version, the tree must include all vertices, but its edges do not need to be present in the underlying graph. This variant arises in the context of multi-commodity tree-based routing [29, 13], and appears to be computationally easier that STC, as it admits an $O(\log n)$ approximation.

Readers interested in learning more about the STC problem are referred to the survey by Otachi [26] that covers the state-of-the-art as of 2020, and to the recent paper by Lampis et al. [18] that has additional information about some recent work, in particular about the parametrized complexity of STC.

2 Preliminaries

Throughout the paper, by $G = (V, E)$ we denote an undirected graph with vertex set V and edge set E. For a vertex u, by E_u we denote the set of edges incident with u and by N_u or $N(u)$ we denote the set of u's neighbors. We extend this notation naturally to sets of vertices. For a set of edges $E' \subseteq E$, $V(E')$ denotes the set of all vertices incident with some edge of E'.

Recall that a connected graph $G = (V, E)$ is said to be *K-edge-connected* if it remains connected even after removing $K - 1$ edges. For any subset $X \notin \{\emptyset, V\}$ of vertices, by ∂X (or $\partial \bar{X}$) we denote the set of edges between X and $\bar{X} = V \setminus X$, and we call it a *cut*. We refer to X and $\bar{X}$ as the *shores* of cut ∂X. If $|\partial X| = K$, we say that ∂X is a *K-cut*, and if $|X| = 1$ or $|\bar{X}| = 1$ then we call cut ∂X *trivial*.

If T is a spanning tree of G and e is an edge of T, removing e from T disconnects T into two connected components. Given a vertex x of T, we denote the component containing x by T_e^{+x} and the component not containing x by T_e^{-x} (we interpret these as sets of vertices of G). The cut $\partial T_e^{+x} = \partial T_e^{-x} = \partial T_e$ is called *the cut induced by e*. (So T_e^{+x} and T_e^{-x} are its two shores.) Its cardinality $|\partial T_e|$ is called the *congestion of e in T* and is denoted by $\mathrm{cng}_{G,T}(e)$, or $\mathrm{cng}_T(e)$ if G is understood from context. The *congestion of tree* T, denoted $\mathrm{cng}(G, T)$, is the maximum edge congestion in T. The minimum value of $\mathrm{cng}(G, T)$ over all spanning trees T of G is called the *spanning-tree congestion of* G and is denoted $\mathrm{stc}(G)$. It is easy to see that this definition, expressed in terms of induced cuts, is equivalent to the definition given at the beginning of Section 1.

3 $\mathbb{NP}$-Hardness for Degree-3 Graphs

In this section we prove Theorem 1.1. Our proof is via a polynomial-time reduction from an $\mathbb{NP}$-complete version of SAT (defined below), mapping a boolean expression ϕ into a graph G and integer K, such that $\mathrm{stc}(G) \leq K$ if and only if ϕ is satisfiable. G consists of multiple *gadget* subgraphs, some corresponding to variables and some to clauses, as well as one additional *root gadget*, with appropriate connections in and between these gadgets. Some gadgets are constructed from smaller *sub-gadgets*. The most basic gadget is called a *double-weight gadget*, and it allows us to use edges that are assigned two weight values, with appropriate interpretation. Using double-weighted edges, we construct a more complex *flower gadget* that will be used as the root gadget and as the gadgets for some clauses. The restriction to degree 3 makes these constructions quite intricate. A considerably simpler proof for graphs of degree at most 4 can be found in Appendix A.

Problem (M2P1N)-SAT. This is an $\mathbb{NP}$-complete restriction of SAT [23], whose instance is a boolean expression in conjunctive normal form with the following properties:

(i) each clause contains either three positive literals (3P-clause), or two positive literals (2P-clause), or two negative literals (2N-clause), and

(ii) each variable appears exactly three times, exactly once in each type of clause, and any two clauses share at most one variable.

We use the following conventions: letter ϕ is an instance of (M2P1N)-SAT, boolean variables are denoted with Latin letters x, y, z, while for clauses we use Greek letters κ, α, β, γ and π. Typically, α, β and γ denote a 2N-clause, 3P-clause and 2P-clause, respectively, κ is a clause of any type, and π is a positive clause. For a variable x, by the *2N-clause of* x we mean the unique 2N-clause that contains the negative literal of x. Similarly, the *2P-* and *3P-clauses of* x are the unique 2P- and 3P-clauses, respectively, that contain the positive literal of x.

Double weights. In the graph we construct from an instance of (M2P1N)-SAT we will need the notion of double-weighted edges, introduced by Luu and Chrobak [23]. Consider a pair of weight functions $\mathrm{w}_1, \mathrm{w}_2 : E \to \{1, ..., M\}$, where M is a positive integer whose value is polynomial in n. Given a spanning tree T and an edge $e = (u, v) \in T$, we define the *weighted congestion* of e to be

$$\mathrm{cng}_{G,T}(e) \ = \ \mathrm{w}_2(e) + \textstyle\sum_{e' \in \partial T_e \setminus \{e\}} \mathrm{w}_1(e') \ .$$

The definitions of $\mathrm{cng}(G, T)$ and $\mathrm{stc}(G)$ extend naturally to double-weighted graphs. We also define the *weighted degree* of v to be $\deg_G(v) = \sum_{(u,v) \in E} \mathrm{w}_1(u, v)$. (Note that it depends only on weight function w_1.) As long as the meaning is clear from context, we will often drop the word *weighted* and write simply *congestion* or *degree*, while we mean the weighted versions of these terms.

When $\mathrm{w}_1(e) = \mathrm{w}_2(e) = 1$, we say that e is *unweighted*. For a double-weighted edge e with $(a, b) = (\mathrm{w}_1(e), \mathrm{w}_2(e))$, we write its weight as $a\!:\!b$.

We will only use weight functions such that $\mathrm{w}_1(e) \leq \mathrm{w}_2(e)$ for each $e \in E$, so one can think of w_1 as the *light weight* function and w_2 as the *heavy weight* function. An important intuition is that the edges in T contribute their heavy weight to their induced cuts, while other edges contribute their light weight to the cuts they belong to. This is why using double-weight edges gives us more control over the structure of trees with optimal congestion, thus greatly simplifying the construction of our target graph.

An edge $e = (u, v)$ with $\mathrm{w}_1(e) = \mathrm{w}_2(e)$ can be replaced by $\mathrm{w}_1(e)$ edge-disjoint paths from u to v without affecting the congestion or vertex degrees. As shown by Luu and Chrobak [23], this idea can be extended to *unequal* double weights: in a graph G, an edge with weight $a\!:\!b$ (under some mild assumptions) can be replaced by an appropriate *double-weight gadget*, so that the resulting graph G' has the property that $\mathrm{stc}(G) \leq K$ if and only if $\mathrm{stc}(G') \leq K$. In their construction, the maximum degree of G' becomes large if b is large. Lampis et al. [18] provide a low-degree double-weight gadget, but it is not sufficient for our purpose (because it contains vertices of degree 4).

The construction of our degree-3 double-weight gadget $\mathcal{W}(a, b)$, where a, b represent a double weight $a : b$, is given in Appendix B. There, we also prove (see Lemma B.1) that for $a < b$ and $b - a \leq K - 2$, $\mathcal{W}(a, b)$ has the following property: if the maximum degree in G is $\Delta \geq 3$ and G' is the graph obtained from G by replacing an edge with weight $a\!:\!b$ by a copy of $\mathcal{W}(a,b)$ then (i) the naximum degree in G' is also Δ and (ii) $\mathrm{stc}(G) \le K$ if and only if $\mathrm{stc}(G') \le K$.

Flowers. One can think of our construction as having an intermediate implicit step, where variables and clauses are mapped into vertices of an auxiliary graph G^* that also contains a high-degree *root* vertex. To obtain the final graph G, the high-degree vertices are replaced by other appropriate gadgets. One gadget is called a *flower*, and it will be used to replace 2N-clause vertices and the root vertex of G^*. One other (unnamed) gadget, with a slightly different functionality, will be used as the variable gadget.

An *ℓ-terminal flower gadget with integrality* K, denoted $\mathcal{F}(\ell, K)$, is the following graph:

- It has 3ℓ vertices $c_1, \ldots, c_\ell$, $d_1, \ldots, d_\ell$, and $t_1, \ldots, t_\ell$, that we call the *core*, *dummy*, and *terminal* vertices, respectively. Vertex c_1 is special and is designated as the *center* of the flower.
- For each $i \in [\ell]$[1], it has the following edges (indexing is cyclic, so $\ell + 1 = 1$): (i) (c_i, c_{i+1}), (t_i, d_i) and (t_i, d_{i+1}), all with weight 1, (ii) (c_i, d_i) with weight $1\!:\!K-1$ for $i \ne 1$ and weight 1 for $i = 1$.

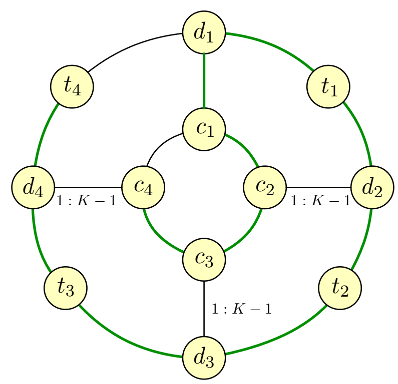

Figure 1: A 4-terminal flower $\mathcal{F}(4, K)$, and its congestion-5 spanning tree marked with thick (green) lines.

Figure 1 shows $\mathcal{F}(4, K)$. Notice that the core and dummy vertices have degree 3, while the terminal vertices have degree 2 (these will be used to attach the flower to the rest of the graph via edges with unit w_1-weights). $\mathcal{F}(\ell, K)$ has $O(\ell)$ vertices and edges, and it has a spanning tree with congestion $\ell + 1$ given by edges (c_i, c_{i+1}), (t_i, d_i), (t_i, d_{i+1}), for $i \in [\ell - 1]$, and (t_ℓ, d_ℓ), (d_1, c_1).

Expanding on the intuition outlined earlier, to simulate a high-degree vertex, a flower should ideally have the property that a spanning tree with congestion K can visit it only once in the sense that the flower lies wholly inside one shore of any cut induced by any non-flower edge in this tree. As this is difficult to achieve using degree-3 vertices, we instead require only the core to lie on one shore of such cuts. To make up for this relaxation, the flower has the property that disjoint paths touching the gadget's terminals can be extended to disjoint paths touching the core. This disjoint-paths property will be crucial in the proof.

Let G be a double-weighted graph, and $U \subsetneq V$ a set of vertices that induces a flower subgraph $\mathcal{F} = \mathcal{F}(\ell, K) = (U, F)$ such that the cut ∂U consists of exactly ℓ edges, each connected to a different terminal in $\mathcal{F}$. The intuition above is formalized below.

Observation 3.1. *For any spanning tree T of G with $\mathrm{cng}_G(T) \le K$ and an edge $e \in T \setminus F$, all the core vertices of $\mathcal{F}$ belong to the same shore of the cut ∂T_e.*

[1] By $[\ell]$ we denote the set $\{1, 2, \ldots, \ell\}$.

Proof. Observe first that T contains no edges between core and dummy vertices of $\mathcal{F}$ except (d_1, c_1): if $(d_i, c_i) \in T$ for some $i \neq 1$, then the cut induced by (d_i, c_i) would have congestion $K + 1$ (because (d_i, c_i) itself contributes $K - 1$, and there are two edge-disjoint paths from c_i to d_i within $\mathcal{F}$ not containing (d_i, c_i)), contradicting the assumption that $\mathrm{cng}_G(T) \leq K$. Thus, T contains all but one edge from the core $c_1 - c_2 - ... - c_\ell - c_1$, which implies the observation. □

Note that in the flower $\mathcal{F}(\ell, K)$, the ℓ length-2 paths $t_i - d_i - c_i$ are disjoint. Thus, given a collection of edge-disjoint paths $\{P_i\}_{i \in I}$ in G indexed by $I \subseteq [\ell]$, where each path P_i starts at some u_i in $V - U$, ends at terminal t_i of $\mathcal{F}$ and does not contain any other vertex of $\mathcal{F}$, we can easily extend it to a collection $\{P'_i\}_{i \in I}$ of edge-disjoint paths with each P'_i starting at u_i and ending at c_i.

The reduction. Given an instance ϕ of (M2P1N)-SAT, we convert it into a double-weighted graph G with maximum degree 3, and a constant K such that:

($*$) the boolean expression ϕ is satisfiable if and only if $\mathrm{stc}(G) \leq K$.

The weights in G will be bounded by a polynomial function of the size of ϕ; thus, per Lemma B.1 and the construction of the double-weight gadget $\mathcal{W}(a, b)$ in Appendix B, this reduction will be sufficient to establish Theorem 1.1.

Let $K \geq 6$ be a positive integer to be specified later, and let n, m, m_1 and m_2, resp., be the number of variables, clauses, 2N-clauses and 2P-clauses of ϕ, resp. We construct a double-weighted graph $G = (V, E; \mathrm{w}_1, \mathrm{w}_2)$ as follows (cf. Fig. 2):

- Create a copy $\mathcal{R}$ of $\mathcal{F}(2m_1 + m_2 + n, K)$, that we call *the root* or *root-gadget*: it has one terminal $t^x_{\mathcal{R}}$ per each variable x, one terminal $t^\gamma_{\mathcal{R}}$ per each 2P-clause γ, and two terminals $t^{\alpha,1}_{\mathcal{R}}$ and $t^{\alpha,2}_{\mathcal{R}}$ per each 2N-clause α. (How exactly we assign terminals to clauses/variables is irrelevant.)
- For each positive clause π, create a vertex π. For every 2N-clause α, create a flower $\mathcal{H}_\alpha = \mathcal{F}(4, K)$, called the *$\alpha$-gadget* or *clause-gadget*, and denote its terminals by $t^{\mathcal{R},1}_\alpha, t^{\mathcal{R},2}_\alpha, t^x_\alpha, t^y_\alpha$, where x and y are the variables in α. (How we assign labels to terminals is irrelevant).
- For each variable x, create a length-4 cycle $x^{2\mathrm{N}} - x^{2\mathrm{P}} - x^{\mathcal{R}} - x^{3\mathrm{P}} - x^{2\mathrm{N}}$ with unweighted edges, called a *variable-gadget* or *x-gadget*. (The order of vertex labels in this cycle *is important*.) Then add a *root-variable edge* $(x^{\mathcal{R}}, t^x_{\mathcal{R}})$ with weight $1{:}K-5$, and *clause-variable* unweighted edges $(t^x_\alpha, x^{2\mathrm{N}})$, $(\beta, x^{3\mathrm{P}})$, $(\gamma, x^{2\mathrm{P}})$, where α, β, γ are the 2N-clause, 3P-clause, and 2P-clause of x, respectively.
- For each 2P-clause γ, add one *root-clause* edge $(\gamma, t^\gamma_{\mathcal{R}})$ with weight $1{:}K-1$. For each 2N-clause α, add two *root-clause* edges $(t^{\mathcal{R},1}_\alpha, t^{\alpha,1}_{\mathcal{R}})$ and $(t^{\mathcal{R},2}_\alpha, t^{\alpha,2}_{\mathcal{R}})$, each of weight $1{:}K-1$.

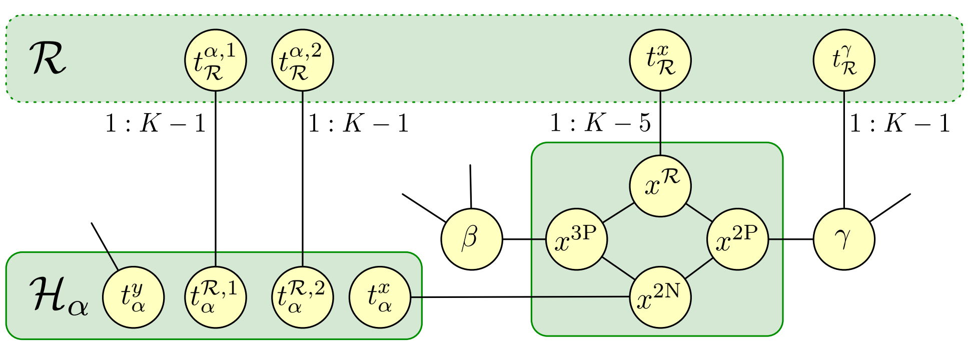

Figure 2: The structure of G.

By inspection, all vertices in this construction have degree 3, and $\mathrm{w}_1(e) = 1$ for all $e \in E$. The number of edges $|E|$ in G is independent of K, so we can set $K = 2|E|$. Our double-weighted STC instance G is now fully specified.

Any edge belonging to a variable, clause, or root gadget is called an *internal edge*, while every other edge is *external*. We identify flowers by the corresponding clause (or root); for example by *core of clause* α we mean the core of $\mathcal{H}_\alpha$. (Similar terminology applies to flower terminals and centers.) For the sake of uniformity, we also refer to positive-clause vertices as *centers*. In the proof it will be convenient to occasionally work in the auxiliary multi-graph G^* given by contracting all internal edges of G, with each flower and variable gadget contracted to a single vertex, for which we use the same notation as the gadget itself. We extend this convention to the edges in G^*: for example a root-clause edge $(t_\alpha^{\mathcal{R},1}, t_{\mathcal{R}}^{\alpha,1})$ in G between a terminal of clause α and a terminal of the root is represented by edge $(\alpha, \mathcal{R})$ in G^* between $\mathcal{R}$ and α.

Correctness. We now need to prove that our construction is correct, namely that it satisfies the condition $(*)$. We prove the two implications in $(*)$ separately.

$(\Rightarrow)$ Given a satisfying assignment for ϕ, we convert it into a spanning tree T for G as follows: (1) for each gadget, add a spanning-tree of unweighted edges for this gadget to T (one always exists for flowers and 4-cycles), (2) add all root-variable edges, (3) for each clause κ, pick any (exactly one) variable x whose assignment satisfies κ and add the clause-variable edge from κ to x to T.

It is easy to see that T is a spanning tree for G. Furthermore, we can obtain a spanning tree T^* for G^* by contracting the internal edges in G, and every clause is a leaf in T^*. We now show that $\mathrm{cng}_{G,T}(e) \le K$ for all $e \in T$.

When $e \in T$ is unweighted, we trivially have $\mathrm{cng}_{G,T}(e) \le |E| \le K$. Otherwise $e = (x, t_{\mathcal{R}}^x) \in T$ is an external root-variable edge, in which case the vertices for any given gadget will be together on one shore of the cut induced by e, implying $\mathrm{cng}_{G,T}(e) = \mathrm{cng}_{G^*,T^*}(x, \mathcal{R})$. Thus, it suffices to deal with congestion in G^*.

Let α, β, γ be the 2N-clause, 3P-clause, and 2P-clause containing x respectively. One shore of the cut induced by $(x, \mathcal{R})$ is given either by (i) $\{x\}$, (ii) $\{\kappa, x\}$ for $\kappa \in \{\alpha, \beta, \gamma\}$, or (iii) $\{x, \beta, \gamma\}$. In case (i), $\mathrm{cng}_{G^*,T^*}(x, \mathcal{R}) = K - 5 + \deg_{G^*}(x) - 1 \le K - 2$. In case (ii), $\mathrm{cng}_{G^*,T^*}(x, \mathcal{R}) = K - 5 + 3 + 2 \le K$, since κ is incident to at most 3 edges in the cut and x is incident to 2 edges besides $(\mathcal{R}, x)$. In case (iii), $\mathrm{cng}_{G^*,T^*}(x, \mathcal{R}) = K - 5 + 2 + 2 + 1 = K$, since both positive clauses are incident to 2 cut edges while x is incident to 1 besides $(\mathcal{R}, x)$.

$(\Leftarrow)$ Given a spanning tree T for G with $\mathrm{cng}(G, T) \le K$, our goal is to construct a satisfying assignment for ϕ. Ideally, T would have a form similar to the spanning tree described in the $\Rightarrow$ direction, but this may not be the case. Nevertheless, T has enough structure to define a variable assignment.

First, we give a suitable definition of *connection* between clause-gadgets and variable-gadgets that will be used to define our boolean assignment. For a clause κ containing variable x, a *traversal from* κ *to* x is a path in T that starts from the center of κ and contains exactly two external edges: the first being the clause-variable edge from κ to x (the *entering edge*), and the second being any other external edge adjacent to the x-gadget (the *exiting edge*). We say that T *traverses* x *from* κ when such a path exists. Intuitively, a traversal from κ to x is a natural way to lift the edge (κ, x) from G^* to a path in G. We emphasize that a traversal begins at the *centers* of flowers, as opposed to terminals or dummy nodes; this becomes relevant later in Claim 3.3.

The following lemma is sufficient to construct an assignment for ϕ:

Lemma 3.2. *Tree* T *has the following properties: (a)* T *does not contain any root-clause edges. (b) If* T *traverses* x *from a negative clause, then* T *does not traverse* x *from any positive clause.*

Assuming the lemma above holds, we define our satisfying assignment as follows: for each variable x, make it false if T traverses it from a negative clause, otherwise make it true. Lemma 3.2(a) implies that T includes for each clause κ a traversal from κ to some variable x appearing in it. (In particular, a traversal appears as a prefix of the path in T from κ's center to the center of the root.) x is false when κ is negative by definition of the assignment, and x is true when κ is positive by Lemma 3.2(b). Therefore all clauses are satisfied.

Proof. First we prove Lemma 3.2(a). Suppose for contradiction that T contains some root-clause edge e from a two-literal clause κ. If $\kappa = \alpha$ is a 2N-clause, then $e = (t_\alpha^{\mathcal{R},j}, t_{\mathcal{R}}^{\alpha,j})$, for $j \in \{1,2\}$, and there are two edge-disjoint paths (neither containing e) from $t_\alpha^{\mathcal{R},j}$ to $t_{\mathcal{R}}^{\alpha,j}$: for each variable x of α, walk from α to the x-gadget, and then to the root. (By the structure of flowers, discussed earlier, we can ensure that these two paths are indeed edge-disjoint.) This implies that $\mathrm{cng}_{G,T}(e) \geq \mathrm{w}_2(e) + 2 = K - 1 + 2 = K + 1$, contradicting $\mathrm{cng}_G(T) \leq K$. A similar reasoning applies when κ is a 2P-clause.

Next we prove Lemma 3.2(b). Let x be a variable and α, β, γ be the 2N-clause, 3P-clause, and 2P-clause of x, respectively. Assume for contradiction that T traverses x from α and at least one positive clause β or γ. We show that this assumption implies that $\mathrm{cng}_G(T) > K$. First we prove the following claim.

Claim 3.3. *For some clause $\pi \in \{\beta, \gamma\}$, the path in T from the center of α to π contains exactly two external edges, the first being the clause-variable edge $\left(t_\alpha^x, x^{2\mathrm{N}}\right)$ from α to x, and the second being the clause-variable edge $\left(\gamma, x^{2\mathrm{P}}\right)$ or $\left(\beta, x^{3\mathrm{P}}\right)$ from x to π.*

To justify Claim 3.3, let P be a traversal from α to x. If the exiting edge of P is $\left(x^{2\mathrm{P}}, \gamma\right)$ (resp. $\left(x^{3\mathrm{P}}, \beta\right)$), then P is simply the path in T from the center of α to γ (resp. β), and the claim is satisfied for $\pi = \gamma$ (resp. β). Otherwise, the exiting edge of P is $\left(x^{\mathcal{R}}, t_{\mathcal{R}}^x\right)$, which of course implies that $x^{2\mathrm{N}}$ and $x^{\mathcal{R}}$ are both in P. Now let $\pi \in \{\beta, \gamma\}$ be a clause where T traverses x from π, and let P' be a corresponding traversal. Then in P', the entering edge must be succeeded by an edge containing $v \in \left\{x^{2\mathrm{N}}, x^{\mathcal{R}}\right\}$, due to the way variable gadgets are labeled. This implies that P and P' overlap. The path from α's center to π is then given by joining the sub-path in P from the center of α to v, followed by the sub-path in P' from v to π. Claim 3.3 then follows.

Continuing the proof of Lemma 3.2(b), let $\pi \in \{\beta, \gamma\}$ be a clause satisfying Claim 3.3. By Lemma 3.2(a), the path in T from the center of α to the center of the root must contain some root-variable edge; suppose $e = \left(y^{\mathcal{R}}, t_{\mathcal{R}}^y\right)$ is the first such edge on this path. Clearly, the center of the root lies on the shore $T_e^{-y^{\mathcal{R}}}$ of the cut ∂T_e, while the center of α lies on the opposite shore $T_e^{+y^{\mathcal{R}}}$. On the other hand, the path described in Claim 3.3 does not contain any root-variable edges, implying that π and $x^{2\mathrm{N}}$ also lie on shore $T_e^{+y^{\mathcal{R}}}$. In conjunction with Observation 3.1, we obtain:

Corollary 3.4. *The core of the root is on the shore $T_e^{-y^{\mathcal{R}}}$, while $x^{2\mathrm{N}}$, π, and the core of α are on the shore $T_e^{+y^{\mathcal{R}}}$.*

We now show that $\mathrm{cng}_T(e) > K$. The definition of (M2P1N)-SAT implies that there are four distinct variables z_1, z_2, z_3, z_4, none equal to x, such that $z_1 \in \alpha$, $z_2, z_3 \in \beta$, and $z_4 \in \gamma$. This in turn implies that there are 7 edge-disjoint paths crossing ∂T_e. In particular, each path begins at either $x^{2\mathrm{N}}$, π, or a core vertex of α, then ends at a core vertex of the root. As explained in the discussion of flower gadgets, it is sufficient to specify terminals as path endpoints. Three of these paths are

$$t_\alpha^{\mathcal{R},1} - t_{\mathcal{R}}^{\alpha,1}, \qquad t_\alpha^{\mathcal{R},2} - t_{\mathcal{R}}^{\alpha,2}, \qquad t_\alpha^{z_1} - z_1^{2\mathrm{N}} - z_1^{2\mathrm{P}} - z_1^{\mathcal{R}} - t_{\mathcal{R}}^{z_1}.$$

The choice of the four remaining paths depends on whether $\pi = \beta$ or γ. If $\pi = \beta$, these paths are

$$x^{2\mathrm{N}} - x^{3\mathrm{P}} - x^{\mathcal{R}} - t^{x}_{\mathcal{R}}, \qquad \beta - z_2^{3\mathrm{P}} - z_2^{\mathcal{R}} - t^{z_2}_{\mathcal{R}},$$
$$\beta - z_3^{3\mathrm{P}} - z_3^{\mathcal{R}} - t^{z_3}_{\mathcal{R}}, \qquad x^{2\mathrm{N}} - x^{2\mathrm{P}} - \gamma - t^{\gamma}_{\mathcal{R}}.$$

If $\pi = \gamma$, these paths are

$$x^{2\mathrm{N}} - x^{2\mathrm{P}} - x^{\mathcal{R}} - t^{x}_{\mathcal{R}}, \qquad \gamma - t^{\gamma}_{\mathcal{R}},$$
$$\gamma - z_4^{2\mathrm{P}} - z_4^{\mathcal{R}} - t^{z_4}_{\mathcal{R}}, \qquad x^{2\mathrm{N}} - x^{3\mathrm{P}} - \beta - z_2^{3\mathrm{P}} - z_2^{\mathcal{R}} - t^{z_2}_{\mathcal{R}}.$$

At most one of these paths contains e, implying $\mathrm{cng}_{G,T}(e) \geq K - 5 + 6 = K + 1$, contradicting $\mathrm{cng}(G,T) \leq K$. This completes the proof of Lemma 3.2. □

4 Cactus Representation and Spanning-Tree Congestion

In this section, laying the groundwork for our algorithm in Section 5, we analyze the structure of K-edge-connected graphs whose spanning-tree congestion is K. We start, in Section 4.1, by reviewing the properties of K-edge-connected graphs, captured by so-called *cactus representation*. Then, in Section 4.2, we focus on the special case of graphs with congestion K, and we prove that this congestion assumption implies additional structural properties of such graphs, that will lead to an efficient algorithm.

Throughout this section we assume that $G = (V, E)$ is a given K-edge-connected graph with $n = |V|$ vertices and $m = |E|$ edges.

4.1 Cactus Representation

Two cuts ∂X and ∂Y are called *nested* if one of the four pair-wise shore intersections $X \cap Y$, $\bar{X} \cap Y$, $X \cap \bar{Y}$ and $\bar{X} \cap \bar{Y}$ is empty. If all four intersections are non-empty then we say that cuts ∂X and ∂Y *cross*. A family of cuts is called *laminar* if any two cuts in it are nested. By $\partial_Y X$ we denote the subset of cut ∂X consisting of edges whose both endpoints are in Y.

Theorem 4.1. [6]. *Let G be a K-edge-connected (multi-)graph.*
(a) *If K is odd, then G has no crossing K-cuts. That is, the family of K-cuts is laminar.*
(b) *If K is even, then any two crossing K-cuts ∂X, ∂Y in G satisfy $|\partial_Y X| = |\partial_{\bar{Y}} X| = |\partial_Y \bar{X}| = |\partial_{\bar{Y}} \bar{X}| = K/2$. There are no edges between $X \cap Y$ and $\bar{X} \cap \bar{Y}$, and between $X \cap \bar{Y}$ and $\bar{X} \cap Y$.*

This theorem can be refined to produce an even more informative representation of K-cuts, in terms of so-called *cactus graphs*. For cactus graphs, we will use terminology of *nodes* and *links* (instead of vertices and edges). A connected multigraph is called a *cactus graph* if every link belongs to exactly one cycle. (Equivalently, it is a 2-edge-connected graph whose biconnected components are cycles.) Cactus cycles of length 2 are called *trivial*. A degree-2 node of a cactus is called an *external node*, and any other node is called an *internal node*. From the definition, it follows that each cactus has at least one external node. If all its cycles are trivial, the cactus forms a tree whose adjacent nodes are connected by two parallel links and the external nodes are its leaves.

Theorem 4.2. [6]. *Let $G = (V, E)$ be a K-edge-connected graph. Then there is a cactus graph $\mathfrak{C}_G = (U, F)$ and an associated mapping $\phi : V \to U$ with the following properties:*
(a) *For each set $X \subseteq V$, ∂X is a K-cut if and only if $X = \phi^{-1}(Q)$ for some $Q \subseteq U$ such that ∂Q is a 2-cut of $\mathfrak{C}_G$.*

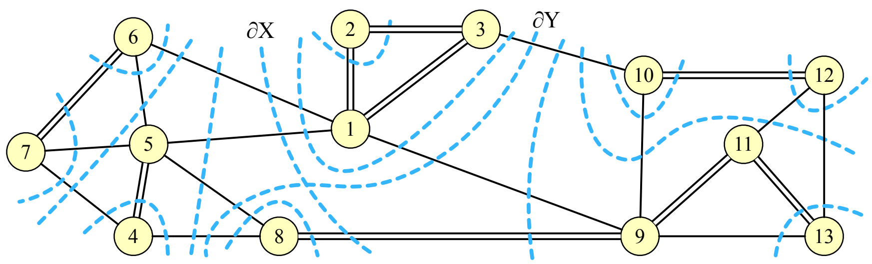

Figure 3: A 4-edge-connected multigraph and all its 4-cuts. Parallel lines represent double parallel edges. Each vertex v_i is identified by its index i. For $X = \{v_4, v_5, v_6, v_7, v_8\}$ and $Y = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$, cuts ∂X, ∂Y cross.

(b) *If K is odd, then all cycles in $\mathfrak{C}_G$ are trivial; that is, $\mathfrak{C}_G$ is a tree with adjacent nodes connected by two parallel links.*

The pair $\mathfrak{C}_G, \phi$ is called a *cactus representation* of G (see Figures 3 and 4). A cactus representation of G of size $O(n)$ can be computed in near-linear time $\tilde{O}(m)$ [12].

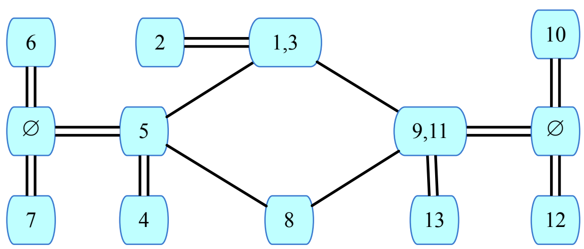

Figure 4: A cactus representation of the graph from Figure 3. The numbers represent indices of vertices in G that are mapped by ϕ to the corresponding node in $\mathfrak{C}_G$. Note that the pre-image of a node in $\mathfrak{C}_G$ could be empty.

Basic K-cuts. Per Theorem 4.2, each K-cut of G is represented by a 2-cut of $\mathfrak{C}_G$. Each 2-cut of $\mathfrak{C}_G$ consists of two links that belong to the same cycle. Thus, a cycle of length ℓ in $\mathfrak{C}_G$ represents $\binom{\ell}{2}$ K-cuts of G. Two K-cuts cross in G if and only if they are represented by two crossing 2-cuts of $\mathfrak{C}_G$; that is, the four links in these two 2-cuts belong to the same cycle and alternate in the order around this cycle.

A shore of any K-cut represented by two non-adjacent links of this cycle can be obtained as a union of shores of K-cuts represented by pairs of its consecutive links. For our purpose it is sufficient for us to focus on this subset of ℓ cuts represented by such link pairs. This motivates the following definition.

A K-cut ∂X in G is called a *basic K-cut* if it is represented by a pair of links of $\mathfrak{C}_G$ that share a node. If this shared node is b, we say that ∂X is a *basic K-cut associated with* b. Note that K-cuts represented by two parallel links are also basic and are associated with both endpoints of these links.

From now on, as a rule, when talking about a cut ∂X associated with b, we will represent it by $X = \phi^{-1}(Q)$ for the shore Q in the 2-cut of $\mathfrak{C}_G$ that does *not* contain b.

For each node $b \in \mathfrak{C}_G$ of degree $2d$, the links incident with b form d disjoint pairs with each pair on the same cycle of $\mathfrak{C}_G$. All the basic K-cuts associated with b are then given by these pairs.

Representing them by $\partial Z_1, \ldots, \partial Z_d$ as described above, all the sets $Z_1, \ldots, Z_d$ form a disjoint partition of the set $V \setminus \phi^{-1}(b)$.

Basic K-cuts form a laminar family. In fact, Theorem 4.1 implies even the following two stronger properties that will be crucial for our algorithm.

Observation 4.3. (a) *A basic K-cut does not cross any other K-cut (even a non-basic one).*
(b) *Let $\partial Z_1, \ldots, \partial Z_d$ be all basic K-cuts associated with $b \in \mathfrak{C}_G$. Then for any K-cut ∂Y, there exists j for which either $Y \subseteq Z_j$ or $\bar{Y} \subseteq Z_j$.*

4.2 Cactus Representation for Graphs with Congestion K

We now assume that $\mathrm{stc}(G) \leq K$, and we show that this assumption implies additional properties of G's cactus representation. These properties will play a critical role in our algorithm.

By $\widetilde{T}$ we denote a spanning tree of G with $\mathrm{cng}_G(\widetilde{T}) \leq K$. Note that by the assumption about K-edge-connectivity, $|E_u| \geq K$ for each vertex u, and $\mathrm{stc}(G) = K$. So each edge e of $\widetilde{T}$ induces a K-cut, i.e., we have $|\partial \widetilde{T}_e| = K$. In particular, every leaf u of $\widetilde{T}$ has degree exactly K in G and $E_u = \partial\{u\}$ is a (trivial) K-cut.

Observation 4.4. *For every node $b \in \mathfrak{C}_G$, we have $|\phi^{-1}(b)| \leq 1$.*

Proof. Suppose that $\phi^{-1}(b)$ contains two distinct vertices u and v. Then none of the K-cuts represented by $\mathfrak{C}_G$ separates u from v. On the other hand, any edge on the u-to-v path in $\widetilde{T}$ induces a K-cut separating u and v, a contradiction. □

By Observation 4.4, each node b of $\mathfrak{C}_G$ can be classified into one of the two categories: either $\phi^{-1}(b) = \emptyset$, in which case we refer to b as a *Type*-0 *node*, or $|\phi^{-1}(b)| = 1$, in which case we call it a *Type*-1 *node*. This is refined further in the observation below that follows directly from the fact that if $\deg(u) = K$ then $E_u = \partial\{u\}$ is a trivial K-cut and $\phi(u)$ is a Type-1 external node of $\mathfrak{C}_G$.

Observation 4.5. *Every node $b \in \mathfrak{C}_G$ is of one of the following three types:*
(i) *an external Type-1 node with $b = \phi(u)$ for a vertex u of degree exactly K,*
(ii) *an internal Type-1 node with $b = \phi(u)$ for a vertex u of degree strictly greater than K, or*
(iii) *an internal Type-0 node.*

Basic K-cuts associated with cactus nodes. A key property of $\widetilde{T}$ that we use in our algorithm is that for any basic K-cut ∂X, all edges in $\widetilde{T} \cap \partial X$ share an endpoint w. Lemmas 4.6 and 4.7 below show an even stronger property, namely for any node b, for all basic cuts associated with b, this common endpoint will be the same; furthermore for Type-1 node b, we have $\phi(w) = b$, i.e., we have only this single choice for the common endpoint w. See Figure 5 for an illustration.

Lemma 4.6. *Let b be a Type-1 node and $w \in V$ such that $\phi(w) = b$. Let $\partial Z_1, \ldots, \partial Z_d$ be all basic K-cuts associated with b. Then $\widetilde{T} \cap \bigcup_{i=1}^{d} \partial Z_i \subseteq E_w$.*

Proof. It is sufficient to prove that for any $i \neq i'$, there is no edge in $\widetilde{T}$ connecting Z_i and $Z_{i'}$. For a contradiction, assume that $u \in Z_i$, $v \in Z_{i'}$, and $(u, v) \in \widetilde{T}$. Now consider the first edge e on the path from w to u in the spanning tree $\widetilde{T}$. Edge e induces a K-cut $\partial Y = \partial \widetilde{T}_e$ with has both u and v in the same shore and w in the other shore. Thus for any j, none of the shores of ∂Y can be contained in some set Z_j. This contradicts Observation 4.3. □

Lemma 4.7. *Let b be a Type-0 node, and let $\partial Z_1, \ldots, \partial Z_d$ be all basic K-cuts associated with b. Then there exists j and $w \in Z_j$ such that $\widetilde{T} \cap \bigcup_{i=1}^{d} \partial Z_i \subseteq E_w$.*

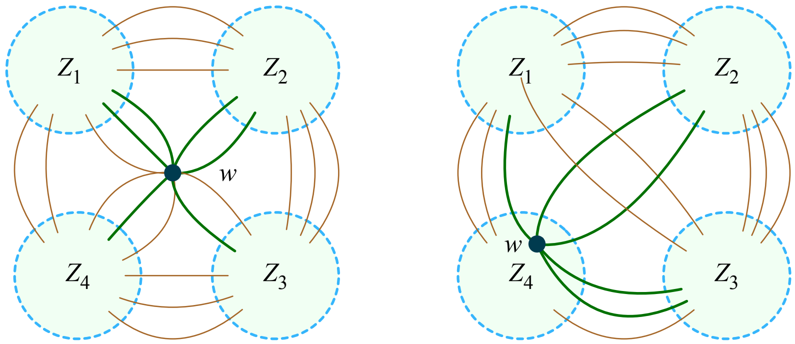

Figure 5: On the left, an illustration of Lemma 4.6. On the right, an illustration of Lemma 4.7. In both examples $K = 8$ and $d = 4$. Edges in $\widetilde{T}$ that cross the basic K-cuts ∂Z_i are green (dark) and thick, non-tree edges are brown (light) and thin.

Proof. Suppose, towards contradiction, that the claim in the lemma is false. That is, there are edges $(u, v), (u', v') \in \widetilde{T} \cap \bigcup_{i=1}^{d} \partial Z_i$ with all endpoints u, v, u', v' distinct. Consider the unique path in $\widetilde{T}$ from $\{u, v\}$ to $\{u', v'\}$ that does not include edges (u, v) and (u', v'). Let e be any edge on this path and let $\partial Y = \partial \widetilde{T}_e$ be the K-cut induced by e. Then ∂Y separates $\{u, v\}$ from $\{u', v'\}$. However, u and v are in two different sets Z_i; the same holds for u' and v'. Thus for any j, none of the shores of ∂Y can be contained in some set Z_j. This contradicts Observation 4.3. $\square$

The corollary below will play a key role in our algorithm, and it follows directly from Lemmas 4.6 and 4.7, as each basic cut is associated with some node b.

Corollary 4.8. *For any basic K-cut ∂Z of G, all edges in $\widetilde{T} \cap \partial Z$ have a common endpoint.*

Basic K-cuts of cactus cycles. Assume now that K is even, and let $C = a_0 - a_1 - \ldots - a_{\ell-1} - a_0$ be a non-trivial cycle in $\mathfrak{C}_G$, so $\ell \geq 3$. We index C cyclically, i.e., $a_i = a_{i \pmod{\ell}}$, for all integers i. For each i, the two consecutive links (a_{i-1}, a_i) and (a_i, a_{i+1}) of C form a 2-cut ∂Q_i of $\mathfrak{C}_G$, where the shore Q_i is chosen so that $a_i \in Q_i$. Let ∂Z_i be the basic K-cut in G represented by the 2-cut ∂Q_i, with $Z_i = \phi^{-1}(Q_i)$. The sets $Z_0, ..., Z_{\ell-1}$ form a partition of V, that is $Z_i \cap Z_{i'} = \emptyset$ for $i \neq i'$ and $\bigcup_{i=0}^{\ell-1} Z_i = V$. By Theorems 4.1 and 4.2, for any i, there are exactly $K/2$ edges between Z_i and Z_{i+1}, i.e., $|\partial Z_i \cap \partial Z_{i+1}| = K/2$; we call this set $\partial Z_i \cap \partial Z_{i+1}$ of $K/2$ edges a *half-K-cut*. It follows that $\partial Z_i \cap \partial Z_{i'} = \emptyset$ if a_i and $a_{i'}$ are not consecutive on C, i.e., whenever $i - i' \notin \{1, -1\} \pmod{\ell}$.

Next, in the lemma below, we show that $\widetilde{T}$ traverses the cuts represented by C in a very restricted way: for some index g, the edges of $\widetilde{T}$ form a length-$(\ell - 1)$ path that traverses consecutive half-K-cuts from Z_g to $Z_{g+\ell-1} = Z_{g-1}$, and the only other edges of $\widetilde{T}$ in the cuts of C are between w_{g+1} and Z_g and between w_{g-2} and Z_{g-1}. In particular, $\widetilde{T}$ has no edges in the half-cut $\partial Z_{g-1} \cap \partial Z_g$. (See Figure 6.)

Lemma 4.9. *There is an index g and vertices $w_g, w_{g+1}, \ldots, w_{g+\ell-1} = w_{g-1}$, with $w_i \in Z_i$ for $i = g, \ldots, g + \ell - 1$, such that the edge set $\widetilde{T} \cap \bigcup_{i=0}^{\ell-1} \partial Z_i$ has the following form:*
(i) *$\widetilde{T} \cap \partial Z_i \cap \partial Z_{i+1} = \{(w_i, w_{i+1})\}$ for $i = g + 1, \ldots, g + \ell - 3$,*
(ii) *$(w_g, w_{g+1}) \in \widetilde{T} \cap \partial Z_g \cap \partial Z_{g+1} \subseteq E_{w_{g+1}}$, $(w_{g-2}, w_{g-1}) \in \widetilde{T} \cap \partial Z_{g-1} \cap \partial Z_{g-2} \subseteq E_{w_{g-2}}$, and*
(iii) *$\widetilde{T} \cap \partial Z_g \cap \partial Z_{g-1} = \emptyset$.*

Proof. First observe that if $\widetilde{T}$ contains two edges in adjacent half-K-cuts, say $e_{i-1} \in \widetilde{T} \cap \partial Z_{i-1} \cap \partial Z_i$ and $e_i \in \widetilde{T} \cap \partial Z_i \cap \partial Z_{i+1}$, then, by Corollary 4.8, they have a common endpoint w_i. Further, since $Z_{i-1} \cap Z_{i+1} = \emptyset$ (and $\ell \geq 3$), we must have $w_i \in Z_i$.

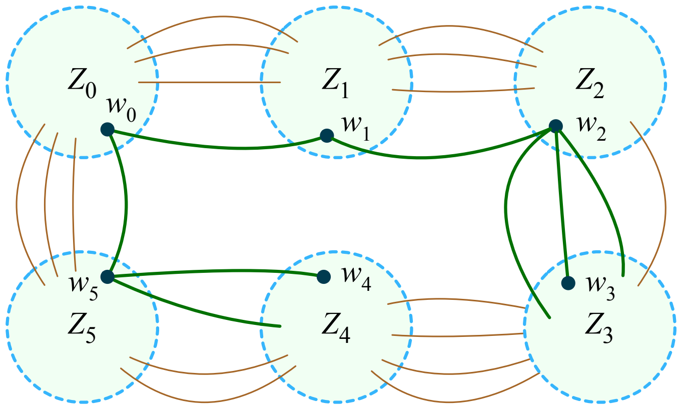

Figure 6: An illustration of Lemma 4.9, for $K = 8$, $\ell = 6$ and $g = 4$.

It is not possible for each half-K-cut $\partial Z_i \cap \partial Z_{i+1}$ to have an edge from $\widetilde{T}$, because then, by the observation from the previous paragraph, these edges would form a cycle. Choose one g for which $\widetilde{T} \cap \partial Z_g \cap \partial Z_{g-1} = \emptyset$. So (iii) is already true. Since $\widetilde{T}$ is spanning, all other half-K-cuts intersect $\widetilde{T}$. Choose one edge from $\widetilde{T} \cap \partial Z_i \cap \partial Z_{i+1}$ for each $i \neq g-1$. Using the observation from the first paragraph, these chosen edges form a path $w_g, \ldots, w_{g+\ell-1}$ with $w_i \in Z_i$ for $i = g, \ldots, g+\ell-1$, and this path satisfies properties (i) and (ii). □

5 A Linear-Time Algorithm

In this section we prove Theorem 1.2, by developing an $\tilde{O}(m)$-time algorithm that, given a K-edge-connected graph G, determines whether $\mathrm{stc}(G) = K$. We start by sketching the basic ideas leading to our algorithm.

A natural attempt to design an algorithm would be to exploit the tree-like structure of cactus graphs, applying the dynamic programming strategy to recursively compute some congestion-related information for the K-cuts of G. The first challenge one encounters is that in general (for even values of K), the K-cuts do not form a laminar structure.

This is where the concept of *basic* K-cuts, introduced in Section 4.1, is helpful. The family of basic K-cuts *is* laminar. More specifically, we will choose an arbitrary degree-K vertex r in G, called the *root* of G (see Section 5.1.) We can then identify each basic K-cut ∂Z by the shore Z that does not contain r. The subset relation between these shores defines a tree structure on the basic K-cuts. The algorithm can then process the basic K-cuts bottom-up in this order.

What information such an algorithm would need to maintain for each basic K-cut ∂Z? A naïve approach would be to somehow keep track of all "candidate crossing-edge sets" $A \subseteq \partial Z$, namely the sets A for which there is a spanning tree $\widetilde{T}$ of G with congestion K such that $\widetilde{T} \cap \partial Z = A$. This raises two issues. One, the resulting algorithm's running time would be exponential in K. Two, obviously, the algorithm does not yet know $\widetilde{T}$ when ∂Z is considered. What's worse, even the congestion of tree edges that are wholly inside shore Z cannot be uniquely determined based only on the subgraph induced by Z. (It may be possible to address the latter issue by storing appropriate information about the tree topology inside Z, but at the price of further increasing the time complexity.)

The characterization of K-edge-connected graphs with congestion K, developed in Section 4, suggests an alternative approach. The key property is Corollary 4.8, which says that if $\widetilde{T}$ is a spanning tree with congestion K then for each basic K-cut ∂Z, all edges in $\widetilde{T} \cap \partial Z$ have a common endpoint. Instead of focussing on tree edges crossing ∂Z, we can instead attempt to recursively compute these common endpoints. This is not quite possible, for the same reason as above: we cannot determine if a vertex is such a common endpoint, for some congestion-K spanning tree, without knowing the whole graph G. Generally, the analysis in Section 4, while it provides crucial ideas, is not sufficient in itself to derive a dynamic-programming algorithm because it is expressed in terms of global properties of G. For this, we need a definition of "candidate common endpoints" that is based only on the information from the already processed subgraph.

To address this, in Section 5.1 we establish analogs of the properties in Section 4 expressed in terms of the rooted versions of $\mathfrak{C}_G$ and G. One key observation is this: for any vertex w and for *any* spanning tree T such that $T \cap \partial Z \subseteq E_w$, the congestion of T's edges within the subgraph induced by $Z \cup \{w\}$ depends only on this subgraph. If all these congestion values are K, we call T *safe*, and we call w a *hub* for ∂Z. (We also require, for technical reasons, that w has at least one edge crossing ∂Z.) If w is a hub for ∂Z, it only means that the information from the subgraph $Z \cup \{w\}$ is not sufficient to eliminate w as a possible common endpoint of the edges from ∂Z that belong to some congestion-K spanning tree of G. (Later, when processing some ancestor K-cut of Z, it may turn out that such a tree does not exist for w.) We prove that the set of hubs of ∂Z, denoted $\widetilde{H}(Z)$, can be determined from the hub sets of ∂Z's children, thus establishing a recurrence relation that drives the dynamic programming algorithm (see Section 5.2).

While it may not be obvious, computing all hub sets $\widetilde{H}(Z)$ is sufficient to determine whether the congestion of G is K. (That is, for any $w \in \widetilde{H}(Z)$ we do not need to keep track of which tree edges from w cross ∂Z, circumventing the issue we mentioned earlier.) The reason is this: once (and if) we reach the root r, the condition on r being a hub for its cut ∂Z (with $Z = V \setminus \{r\}$) is equivalent to G having a spanning tree with congestion K.

5.1 Rooting G and its Cactus Graph

Throughout this sub-section (Section 5.1) we will assume that G is a K-edge-connected graph and that $|\phi(b)| \leq 1$ for every node $b \in \mathfrak{C}_G$, so that Observation 4.5 can be applied. We fix an arbitrary degree-K vertex $r \in V$ as the *root of* G. By convention, from now on, each K-cut ∂X will be represented by the shore X that does not contain r. This naturally imposes a tree-like structure on basic K-cuts, where a basic K-cut ∂X is a *descendant* of a basic K-cut ∂Y if $X \subsetneq Y$. A descendant ∂X is called a *child* of ∂Y (making ∂Y a *parent* of ∂X) if there is no basic K-cut ∂Z such that $X \subsetneq Z \subsetneq Y$. Since the family of basic K-cuts is laminar by Observation 4.3, every cut except $\partial(V \setminus \{r\})$ has a unique parent. (So the descendant relation is indeed a tree.)

To reflect this ordering of basic K-cuts in G's cactus representation, we root $\mathfrak{C}_G$ at the node $\phi(r)$. In our algorithm we will need a few more related concepts:

- A node a of $\mathfrak{C}_G$ is said to be *below* a node b if b is on every path from a to $\phi(r)$.
- A node a is called the *head* of the cycle C in $\mathfrak{C}_G$, and denoted a_C, if $a \in C$ and all the other nodes of C are below a_C.
- For a node $a \neq \phi(r)$, by Q^a we denote the set of nodes consisting of a and all nodes below a. Also, let $W^a = \phi^{-1}(Q^a)$ be the set of vertices of G represented by Q^a.
- For a cycle C in $\mathfrak{C}_G$, we let $Q^C = \bigcup_{b \in C \setminus a_C} Q^b$. That is, Q^C consists of all nodes that are below a_C, excluding a_C itself. Also, let $W^C = \phi^{-1}(Q^C)$.

The cactus structure implies that each cycle has a unique head, so the definition above is valid. Furthermore, each node except for $\phi(r)$ belongs to a unique cycle, which could be trivial or not, where it is a non-head node. On the other hand, a node can be a head of several cycles.

The following observation, that follows directly from the definitions, summarizes the properties of the basic K-cuts and their ordering, and the corresponding properties of $\mathfrak{C}_G$.

Observation 5.1. *The basic K-cuts of G satisfy the following properties:*

(a) *The family of basic K-cuts of G consists exactly of all K-cuts ∂W^b and ∂W^C, for nodes $b \neq \phi(r)$ and non-trivial cycles C in $\mathfrak{C}_G$. Further, this representation is unique; that is all K-cuts ∂W^b and ∂W^C, for nodes $b \neq \phi(r)$ and non-trivial cycles ∂W^C in $\mathfrak{C}_G$, are different.*

(b) *If C is a trivial cycle of $\mathfrak{C}_G$ then $\partial W^C = \partial W^b$, where b is the node on C different from a_C.*

(c) *For nodes a and b of $\mathfrak{C}_G$ distinct from $\phi(r)$, a is below node b if and only if ∂W^a is a descendant of ∂W^b.*

(d) *For a non-trivial cycle C of $\mathfrak{C}_G$, the parent of ∂W^C is ∂W^{a_C}, and the children of ∂W^C are the K-cuts ∂W^b for $b \in C \setminus \{a_C\}$. Further, $W^C = \bigcup_{b\in C\setminus\{a_C\}} W^b$.*

(e) *For a node $b \neq \phi(r)$ of $\mathfrak{C}_G$, if C is the unique cycle where b is a non-head node, the parent of ∂W^b is either W^{a_C}, if C is trivial, or W^C, if C is non-trivial. The children of W^b are all K-cuts W^C, for cycles C for which $a_C = b$. (Note that, by (b), for trivial cycles C these children also have the form W^a, where a is the node on C other than b.) Further, $W^b = \phi^{-1}(b) \cup \bigcup_{C:b=a_C} W^C$.*

Finally, note that $\phi(r)$ is an external node of $\mathfrak{C}_G$, as r has degree K in G. It follows that there is a unique cycle C with $a_C = \phi(r)$. Thus the K-cut $\partial(V \setminus \{r\})$ is equal to W^C. In particular, if C is trivial, this K-cut is W^b, for the node b in C other than $\phi(r)$.

5.1.1 Safe Trees and Hubs

Through the rest of this section, by a *tree in G* we will mean a tree that is a subgraph of G. To simplify notation we will sometimes treat trees in G as sets of vertices and for such a tree T we will write ∂T to mean $\partial V(T)$.

We now define the concepts of *safe trees* and *hubs*, mentioned earlier in the beginning of Section 5. Roughly, a tree in G is considered *safe* if it cannot be eliminated as a possible subtree of $\widetilde{T}$ (a spanning tree with congestion K) based only on the subgraph of G it spans. These trees are used as partial solutions constructed (implicitly) in the algorithm. As the algorithm proceeds, it may expand a safe tree, but only by connecting its root to a vertex in the new, larger safe tree. A safe tree can get discarded, if we discover that it cannot be expanded to a larger safe tree.

To formally define safe trees, we need to extend some concepts from Section 2, defined for spanning trees, to arbitrary trees in G. If T is any tree in G and $e = (u, v) \in T$ is an edge of T, removing e from T disconnects T into two connected components. As before, given a vertex w of T, we denote the component not containing w by T_e^{-w}. As usual, by ∂T_e^{-w} we denote the set of edges between this component and the rest of the graph.

For any tree T in G and any vertex w in T, (T, w) denotes the rooted version of T, with w designated as its root. Such a rooted tree (T, w) is called *safe* if for each edge e of T we have $|\partial T_e^{-w}| = K$. The concept of a safe tree is a natural generalization of the notion of a spanning tree with congestion K, in the following sense:

Observation 5.2. *Let T be a spanning tree in G and w any vertex. Then (T, w) is safe if and only if $\operatorname{cng}(G, T) \leq K$.*

The lemma below generalizes Corollary 4.8 to an arbitrary safe tree.

Lemma 5.3. *Let ∂Z be a basic K-cut and (T, w) a safe tree in G with $T \cap \partial Z \neq \emptyset$. Then all edges of $T \cap \partial Z$ have a common endpoint.*

Proof. For a contradiction, suppose that f and f' are two edges in $T \cap \partial Z$ such that all four of their endpoints are distinct. Consider a path connecting these edges in T and any edge e on this path in-between f and f'. Then the cut ∂T_e^{-w} is a K-cut, by the definition of a safe tree. However, T_e^{-w} contains exactly one of the edges f and f' with its both endpoints, while the endpoints of the other edge are outside of T_e^{-w}. It follows that the cuts ∂Z and ∂T_e^{-w} are crossing, as each of the four endpoints is in a different shore intersection. Since Z is a basic K-cut, this is a contradiction with Observation 4.3.

From the above paragraph, any pair of edges in $T \cap \partial Z$ must share an endpoint. We still need to argue that *all* edges in $T \cap \partial Z$ must share a common endpoint. Indeed, this is trivial when $|T \cap \partial Z| \in \{1, 2\}$. When $|T \cap \partial Z| \geq 3$, fix any three edges in $T \cap \partial Z$, and observe that they must share a common endpoint, say w, because otherwise they would form a 3-cycle. Any other edge in $T \cap \partial Z$, to share an endpoint with each of these three edges, must have w as an endpoint. □

We are now ready to define hubs. Let ∂Z be a basic cut in G. A node $w \in V(\partial Z)$ is called a *hub* for Z, if there exists a spanning tree T of the subgraph of G induced by $Z \cup \{w\}$ such that (T, w) is a safe tree in G. We call this T a *witness tree* for w and ∂Z. If $w \in Z$, then w is called an *in-hub for Z*, otherwise w is an *out-hub for Z*. By $\widetilde{H}(Z)$ we denote the set of all hubs for Z.

5.1.2 Constructing the Hub Sets for Basic K-Cuts

We now develop the properties of hubs for basic K-cuts ∂W^b that are analogous to Lemmas 4.6 and 4.7, but are formulated in terms of safe trees instead of the congestion-K spanning tree $\widetilde{T}$.

The following observations follow directly from the definitions. They show that the safety property of trees is preserved under some operations, like taking sub-trees or combining disjoint trees with a common root.

Observation 5.4. *Let (T, w) be a safe tree, u a vertex of T, and e the first edge on the path in T from u to w. Then (T_e^{-w}, u) is also a safe tree.*

Observation 5.5. *Let (T, w) and (T', w) be two edge-disjoint safe trees with a common root w. Then $(T \cup T', w)$ is a safe tree.*

Observation 5.6. *Let (T, v) be a rooted tree in G satisfying $|\partial T| = K$, and let $(u, v) \in \partial T$. Then $(T \cup \{(u, v)\}, u)$ is a safe tree if and only if (T, v) is a safe tree.*

Let $Z = W^b$ be a basic K-cut, for some node b of $\mathfrak{C}_G$. We now describe how the set of hubs $\widetilde{H}(Z)$ can be computed from the sets of hubs for the children of cut ∂Z. We break it into three cases: when b is an external node, when it is an internal node of Type 1, and when it is an internal node of Type 0.

The case when b is an external node (so it has no children) is simple. Recall that each external node is a Type-1 node with a degree K vertex in its preimage. The set W^b then contains this single vertex and represents a trivial K-cut.

Lemma 5.7. *Let $b \neq \phi(r)$ be an external node of $\mathfrak{C}_G$ (necessarily of Type-1) with $\phi^{-1}(b) = \{v\}$. Then $\widetilde{H}(W^b) = \{v\} \cup N(v)$.*

Proof. The witness tree for v is the trivial tree with v as a single vertex. For any $w \in N(v)$, the witness tree is the tree with a single edge (v, w). There are no other vertices in $V(\partial W^b)$, so the lemma follows. □

If b is an internal node, of any type, we will denote the children of W^b by $Z_1, \ldots, Z_t$. These children are determined as detailed in Observation 5.1(e). As also explained in Observation 5.1(e), the set W^b is a disjoint union of all sets Z_i, plus a singleton $\phi^{-1}(b)$ if b is of Type-1. This, together with Observation 4.3, implies the following key property:

Observation 5.8. *Let b be an internal node of $\mathfrak{C}_G$, and denote by $Z_1, ..., Z_t$ the children of its corresponding K-cut W^b. If ∂X is a K-cut (not necessarily basic) such that $X \subsetneq W^b$ then $X \subseteq Z_i$ for some i.*

Now we examine the case when b is an internal node of Type-1. We show that the only possible in-hub for W^b is the vertex v with $\phi(v) = b$, and that for v to be a hub it must be an out-hub for each Z_i. The only possible out-hubs are neighbors of v outside W^b, providing that v is an in-hub itself. Formally, we have the following lemma.

Lemma 5.9. *Let b be a Type-1 internal node of $\mathfrak{C}_G$, and let v be the (unique) vertex of G satisfying $\phi(v) = b$. Let $Z_1, \ldots, Z_t$ be the children of W^b. Then $v \in V(\partial W^b) \cap \bigcap_{i=1}^t V(\partial Z_i)$ and*

$$\widetilde{H}(W^b) \;=\; \begin{cases} \{v\} \cup (N(v) \setminus W^b) & \text{if } v \in \bigcap_{i=1}^t \widetilde{H}(Z_i) \\ \emptyset & \text{otherwise} \end{cases} \tag{1}$$

Proof. The first condition, namely that v has edges crossing K-cut ∂W^b and all K-cuts ∂Z_i, follows directly from Lemma 4.6.

In the rest of the proof we show that Equation (1) is true. The argument is by considering three types of vertices: vertices in $\bigcup_{i=1}^t Z_i$, vertex v (the only vertex in $W^b \setminus \bigcup_{i=1}^t Z_i$), and vertices in $V \setminus W^b$. The theorem will follow from the three claims established below.

Claim 5.10. $\widetilde{H}(W^b) \cap (\bigcup_{i=1}^t Z_i) = \emptyset$.

To prove this claim, suppose that $w \in \widetilde{H}(W^b) \cap \bigcup_{i=1}^t Z_i$, and let T be its safe tree. Consider an edge e on the path from w to v in T. By the definition of a safe tree, $\partial X = \partial T_e^{-w}$ is a K-cut. But $X \subsetneq V(T) = W^b$ and, since $v \in X$, we also have $X \not\subseteq \bigcup_{i=1}^t Z_i$, contradicting Observation 5.8.

Claim 5.11. $v \in \widetilde{H}(W^b)$ *iff* $v \in \bigcap_{i=1}^t \widetilde{H}(Z_i)$.

We start with the ($\Leftarrow$) implication. Assume that $v \in \bigcap_{i=1}^t \widetilde{H}(Z_i)$. For all $i = 1, ..., t$, consider the witness tree for v and Z_i. By Observation 5.6 their union is a safe tree rooted at v and thus it is a witness tree for v and W^b. Together with $v \in V(\partial W^b)$ this implies that v is an in-hub for W^b.

To prove the ($\Rightarrow$) implication, suppose that $v \in \widetilde{H}(W^b)$ and let T be its safe tree. We will show that v is an out-hub for each Z_i. Indeed, for any edge $e = (v, u_e)$ from v in T, $u_e \in Z_j$ for some j and $V(T_e^{-v}) \subseteq Z_j$, as ∂T_e^{-v} is a K-cut not containing the whole W^b.

Now consider an arbitrary but fixed i, and the tree T' consisting of all edges e from v to Z_i in T plus a union of all corresponding trees T_e^{-v}. This T' spans $Z_i \cup \{v\}$. For each $e = (v, u_e), u_e \in Z_i$, by Observations 5.4 and Observation 5.6, both (T_e^{-v}, u_e) and $(T_e^{-v} \cup \{e\}, v)$ are safe trees. Now T' is the union of all these trees and thus Observation 5.5 implies that T' is a witness tree for v and ∂Z_i and that $w = v$ is an out-hub for Z_i.

Claim 5.12. *Let $w \in V \setminus W^b$. Then $w \in \widetilde{H}(W^b)$ iff $v \in \bigcap_{i=1}^t \widetilde{H}(Z_i)$ and $w \in N(v) \setminus W^b$.*

The ($\Leftarrow$) implication is trivial: if $v \in \bigcap_{i=1}^t \widetilde{H}(Z_i)$ then, by Claim 5.11, v is an in-hub for W^b. Then the assumption that $w \in N(v) \setminus W^b$ implies that w is an out-hub for W^b, directly by definition.

It remains to prove the ($\Rightarrow$) implication. Suppose that $w \in \widetilde{H}(W^b)$ and let T be its witness tree. By the definition of $\widetilde{H}(W^b)$ we have $w \in V(\partial W^b)$.

We argue first that the degree of w in T is 1. Otherwise, if the degree of w in T is at least 2, let e be an edge from w such that $v \in T_e^{-w}$ (such an edge necessarily exists.) Then T_e^{-w} does not span whole W^b as there is another edge from w to W^b in T. Thus $v \in V(T_e^{-w}) \subsetneq W^b$, and ∂T_e^{-w} is a K-cut, a contradiction again with Observation 5.8.

Therefore the degree of w in T is 1. Let this single edge from w be (w, x). Then Observation 5.6 implies that (T, w) is a safe tree if and only if (T_e^{-w}, x) is a safe tree. Then T_e^{-w} witnesses that x is an in-hub for W^b. But Claims 5.10 and 5.11 imply that only v can be an in-hub for W^b, providing that $v \in \bigcap_{i=1}^t \widetilde{H}(Z_i)$, completing the proof of the ($\Rightarrow$) implication in Claim 5.12. □

If b is an internal node of Type 0, W^b may have multiple in-hubs, but, as we show in the lemma below, they still must be common hubs for all Z_i's. The out-hubs of W^b may either be neighbors of such in-hubs, or common out-hubs for all Z_i's.

Lemma 5.13. *Let b be a Type-0 internal node of the cactus. Let $Z_1, \ldots, Z_t$ be the children of W^b. Then*

$$\widetilde{H}(W^b) \;=\; \bar{H} \cup (N(\bar{H} \cap W^b) \setminus W^b), \quad \textit{where } \bar{H} = V(\partial W^b) \cap \textstyle\bigcap_{i=1}^t \widetilde{H}(Z_i). \tag{2}$$

Proof. We first show the ($\supseteq$) inclusion. Consider some $w \in \bar{H}$. Then w is a hub for each set Z_i. Consider the corresponding witness trees T_i for w and Z_i for all $i = 1, \ldots, t$. By Observation 5.6 their union is a safe tree rooted at w and thus it is a witness tree for w and ∂W^a. Together with $w \in V(\partial W^b)$ this implies that w is an in-hub for W^b.

Next, consider some $u \in N(\bar{H} \cap W^b) \setminus W^b$. That is, $u \in V \setminus W^b$, and u is a neighbor of some $w \in \bar{H} \cap W^b$. By the previous paragraph, w is an in-hub for W^b, so Observation 5.6 implies that u is an out-hub for W^b. This completes the proof of the ($\supseteq$) inclusion.

To show the ($\subseteq$) inclusion, fix any $w \in \widetilde{H}(W^b)$, and let T be its witness tree. Recall that T spans $W^b \cup \{w\}$, so any neighbor of w in T is in $W^b = \bigcup_{i=1}^t Z_i$. We need to show that $w \in \bar{H} \cup N(\bar{H} \cap W^b) \setminus W^b$. To this end, we consider the three cases below.

<u>Case 1</u>: $w \in W^b$. We will show that in this case $w \in \bar{H}$. Note that $w \in V(\partial W^b)$ by the definition of a hub, so we need to show that $w \in \bigcap_{i=1}^t \widetilde{H}(Z_i)$.

We start with the following simple observation. For any edge (w, u) in T, ∂T_e^{-w} is a K-cut and $T_e^{-w} \subseteq W^b \setminus \{w\} \subsetneq W^b$, which implies that T_e^{-w} is a subset of one of the sets Z_i.

Now, fix some arbitrary index $i \in \{1, ..., t\}$. It remains to show that $w \in \widetilde{H}(Z_i)$. Let $e_1, \ldots, e_q$ be all the edges in T from w to Z_i. We claim that the subtrees $T_{e_j}^{-w}$, $j = 1, \ldots, q$ cover all the vertices of Z_i, possibly with the exception of w in case when $w \in Z_i$. Indeed, consider any vertex $x \in Z_i \setminus \{w\}$ and the first edge $e' = (w, u')$ on the path from w to x in T. By the previous observation, $u' \in Z_i$, as $T_{e'}^{-w}$ contains $x \in Z_i$. Using Observations 5.4, 5.6, and 5.5, this implies that $\bigcup_{j=1}^q (T_{e_j}^{-w} \cup \{e_j\})$ is a witness tree for w and ∂Z_i, so $w \in \widetilde{H}(Z_i)$, as needed.

<u>Case 2</u>: $w \notin W^b$ and the degree of w in T is at least 2. We will show that in this case $w \in \bar{H}$. The argument is essentially the same as in Case 1. We have that $w \in V(\partial W^b)$, by the definition of a hub, and it remains to show that $w \in \bigcap_{i=1}^t \widetilde{H}(Z_i)$.

For any edge (w, u) in T, ∂T_e^{-w} is a K-cut and $T_e^{-w} \subseteq W^b \setminus \{u'\} \subsetneq W^b$, where u' is any neighbor of w in T other than u. This implies that T_e^{-w} is a subset of one of the sets Z_i — the same property that we had in Case 1.

Following the same argument as in Case 1, for any index i we can obtain the witness tree for w and ∂Z_i by combining the branches of T inside Z_i, showing that $w \in \widetilde{H}(Z_i)$. Since i is arbitrary, we conclude that $w \in \bigcap_{i=1}^t \widetilde{H}(Z_i)$.

<u>Case 3</u>: $w \notin W^b$ and the degree of w in T is 1. In this case, we show that $w \in N(\bar{H} \cap W^b) \setminus W^b$. Let this single edge from w in T be $e = (w, x)$. Then Observation 5.6 implies that (T, w) is a safe tree

if and only if (T_e^{-w}, x) is a safe tree. However, then T_e^{-w} witnesses that x is an in-hub for W^b and we have already shown that this in turn happens if and only if $x \in \bar{H}$. Thus $w \in N(\bar{H} \cap W^b) \setminus W^b$, completing the proof. $\square$

5.1.3 Constructing the Hub Sets for Non-Trivial Cycles

In the previous sub-section we characterized hubs associated with K-cuts W^b, for nodes $b \in \mathfrak{C}_G$. In this sub-section, we assume that K is even and we analyze hubs associated with the other type of basic K-cuts, namely with K-cuts W^C, for non-trivial cycles C of $\mathfrak{C}_G$. The structural properties we establish are analogous to those in Lemma 4.9, where they were based on a congestion-K spanning tree $\widetilde{T}$. Here, we need to show that similar properties can be derived based on safe trees instead. The path of $\widetilde{T}$ that traversed all basic K-cuts associated with C, identified in Lemma 4.9, will be represented here by two sub-paths emanating from the head node a_C of C, one clockwise and the other counter-clockwise on C. These sub-paths will be referred to as the front and back spine of C.

We denote the nodes of C by $a_0, \dots, a_{\ell-1}, a_\ell = a_0$, in the order along C, where $\ell \geq 3$. That is, the links of C are exactly $(a_0, a_1), (a_1, a_2), ..., (a_{\ell-1}, a_0)$. We assume that $a_0 = a_\ell$ is the head node a_C of C. The children of W^C are the sets $Z_i = W^{a_i}$, $i = 1, \dots, \ell - 1$. For convenience, we also use notation $Z_0 = Z_\ell = V(\partial W^C) \setminus W^C$. We stress that this set Z_0 is not a full shore of W^C (unlike the other sets Z_i), it only includes the endpoints of the edges of the cut ∂W^C that are outside W^C. As in Section 4.2, for each $i = 0, ..., \ell - 1$, the set $\partial Z_i \cap \partial Z_{i+1}$ is a half-K-cut in G (contains exactly $K/2$ edges).

We now define the two spines for C, mentioned earlier:

- A *back spine* is a path $w_s, \dots, w_\ell$ in G, with $s \in \{2, \dots, \ell\}$, such that $w_s \in Z_s \cap \widetilde{H}(Z_{s-1}) \cap \widetilde{H}(Z_s)$, $w_i \in Z_i \cap \widetilde{H}(Z_i)$ for each $i = s+1, \dots, \ell-1$, and $w_\ell \in Z_\ell$.
- Symmetrically, a *front spine* is a path $w_0, ..., w_s$ in G, with $s \in \{0, \dots, \ell-2\}$, such that $w_0 \in Z_0$, $w_i \in Z_i \cap \widetilde{H}(Z_i)$ for each $i = 1, \dots, s-1$, and $w_s \in Z_s \cap \widetilde{H}(Z_s) \cap \widetilde{H}(Z_{s+1})$.

Observation 5.14. *If $w_s, \dots, w_\ell$ is a back spine then $w_i \in \widetilde{H}(Z_{i-1})$ for all $i = s, \dots, \ell$. Symmetrically, if $w_0, \dots, w_s$ is a front spine then $w_i \in \widetilde{H}(Z_{i+1})$ for all $i = 0, \dots, s$.*

Proof. Consider the case of a back spine. For $i = s$ the claim is included in the definition of a back spine. For each $i \in \{s+1, ..., \ell\}$, by the definition of back spines, w_{i-1} is an in-hub for Z_{i-1}, so, since we also have $(w_{i-1}, w_i) \in \partial Z_{i-1}$, Observation 5.6 implies that w_i is an out-hub for Z_{i-1}. The case of the front spine is symmetric. $\square$

Lemma 5.15. *Let C be a non-trivial cycle in the cactus $\mathfrak{C}_G$ with vertices $a_0, a_1, \dots, a_{\ell-1}, a_\ell = a_0$, listed in their order around C. Then*

(i) *If $w_2, \dots, w_\ell$ is a back spine then $w_{\ell-1}, w_\ell \in \widetilde{H}(W^C)$.*
(ii) *If $w_0, \dots, w_{\ell-2}$ is a front spine then $w_0, w_1 \in \widetilde{H}(W^C)$.*
(iii) *If $w_0, \dots, w_{g-2}$ is a front spine and $w_{g+1}, \dots, w_\ell = w_0$ is a back spine, for some index $g \in \{2, \dots, \ell-1\}$, then $w_0 \in \widetilde{H}(W^C)$.*
(iv) *$\widetilde{H}(W^C)$ contains only the vertices included in rules (i), (ii) and (iii).*

Proof. (i) Let $w_2, \dots, w_\ell$ be a back spine. Let T_1 be a witness tree for w_2 and Z_1, and, for each $i = 2, \dots, \ell-1$, T_i be a witness tree for w_i and Z_i. These witness trees exist by the definition of a back spine. Let T' be the tree obtained as a union of trees $T_1, ..., T_{\ell-1}$ and edges $(w_1, w_2), ..., (w_{\ell-2}, w_{\ell-1})$, and T'' be the tree obtained by adding edge $(w_{\ell-1}, w_\ell)$ to T'. From Observations 5.5 and 5.6 we obtain that T' is a witness trees for $w_{\ell-1}$ and W^C, and T'' is a witness tree for w_ℓ and W^C. So $w_{\ell-1}, w_\ell \in \widetilde{H}(W^C)$.

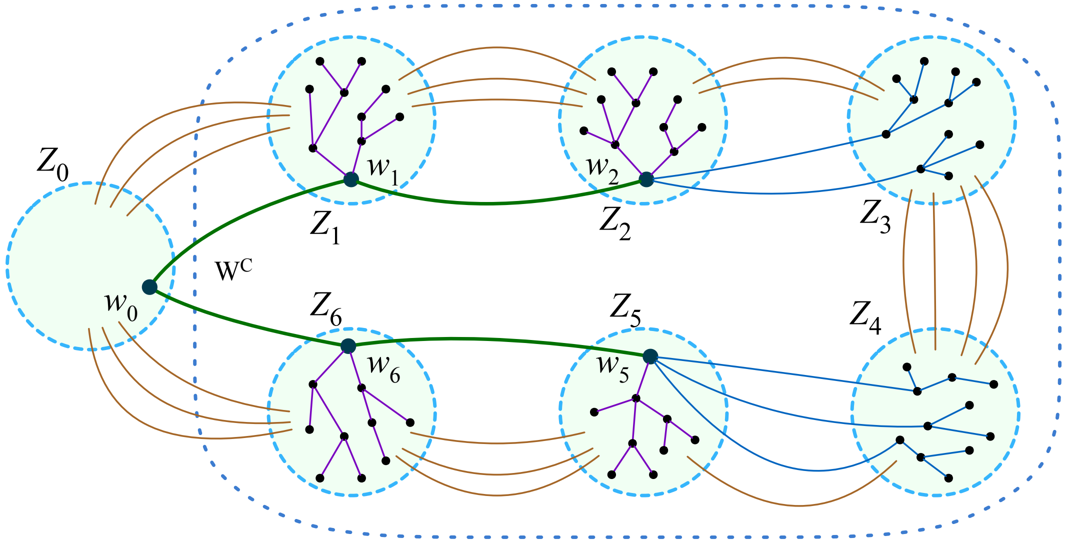

Figure 7: An illustration of Lemma 5.15(iii), for $K = 8$, $\ell = 7$ and $g = 4$. Thick blue edges show the front spine w_0, w_1, w_2 and back spine w_5, w_6, w_7, with $w_7 = w_0$. For $i \in \{1, 2, 5, 6\}$, the witness trees T_i for w_i and Z_i are depicted with thin purple lines. The witness trees for w_2 and Z_3, and for w_5 and Z_4 are depicted with thin blue lines. Thin brown lines are non-tree edges in the cuts ∂Z_i.

The proof of (ii) is symmetric to (i).

(iii) Let $w_0, \ldots, w_{g-2}$ and $w_{g+1}, \ldots, w_{\ell-1}, w_\ell$ be a front and back spine, where $g \in \{2, \ldots, \ell-1\}$. Then, from the definition of these spines, the following witness trees exist:

- a tree T_i for w_i and Z_i, for each $i \in \{1, \ldots, g-2\}$, and a tree T_{g-1} for w_{g-2} and Z_{g-1}, and
- a tree T_i for w_i and Z_i, for each $i \in \{g+1, \ldots, \ell-1\}$ and a tree T_g for w_{g+1} and Z_g.

Let T be a tree obtained as a union of all these trees, along with edges $(w_0, w_1), \ldots, (w_{g-3}, w_{g-2})$ and $(w_{g+1}, w_{g+2}), \ldots, (w_{\ell-1}, w_\ell)$. (See an example in Figure 7.) Using Observations 5.5 and 5.6, T is a witness tree for $w_0 = w_\ell$ and W^C, and thus $w_0 \in \widetilde{H}(W^C)$.

(iv) Assume that $w \in \widetilde{H}(W^C)$, and let T be a witness tree for w and W^C. By the definition of hubs we have $w \in V(\partial W^C)$, so the structure of C implies that $w \in Z_0 \cup Z_1 \cup Z_{\ell-1}$.

We first consider the case when $w \in Z_0 = Z_\ell$, that is w is an out-hub for W^C. The argument relies on the two claims below.

Claim 5.16. *Let $i \in \{1, \ldots, \ell-1\}$. If $T \cap \partial Z_{i-1} \cap \partial Z_i \neq \emptyset$ and $T \cap \partial Z_i \cap \partial Z_{i+1} \neq \emptyset$ then all edges in $T \cap \partial Z_i$ have a common endpoint, say w_i, and $w_i \in Z_i \cap \widetilde{H}(Z_i)$.*

That, under the assumptions of the claim, all edges in $T \cap \partial Z_i$ have a common endpoint, follows from Lemma 5.3. Since $Z_{i-1} \cap Z_{i+1} = \emptyset$, we must have $w_i \in Z_i$.

It remains to show that $w_i \in \widetilde{H}(Z_i)$. Let $e_1 = (w_i, u_1), \ldots, e_k = (w_i, u_k)$ be all edges in T such that $u_j \in Z_i$ for all $j = 1, \ldots, k$. Since $w \notin Z_i$, the first part of the claim implies that for each $x \in Z_i \setminus \{w_i\}$ the path from w to x goes through w_i. So the trees $T_{e_j}^{-w_i}$, for $j = 1, \ldots, k$, along with the singleton $\{w_i\}$, form a partition of Z_i. For each j, by Observations 5.4 and Observation 5.6, both $(T_{e_j}^{-v}, u_j)$ and $(T_{e_j}^{-v} \cup \{e_j\}, v)$ are safe trees. Observation 5.5 now implies that $\bigcup_{j=1}^k (T_{e_j}^{-v} \cup \{e_j\})$ is a witness tree for w_i and Z_i, thus showing that $w_i \in \widetilde{H}(Z_i)$.

Claim 5.17. *Let $i \in \{1, \ldots, \ell-1\}$. Suppose that there is $v \in Z_{i-1} \cup Z_{i+1}$ that is a common endpoint of all all edges in $T \cap \partial Z_i$. Then $v \in \widetilde{H}(Z_i)$.*

The argument is similar to the one above. Let $e_1 = (v, u_1), ..., e_k = (v, u_k)$ be all edges in T such that $u_j \in Z_i$ for all $j = 1, ..., k$. The assumption of the claim implies that for each $x \in Z_i$ the path from w to x goes through v. So the trees $T_{e_j}^{-w_i}$, for $j = 1, ..., k$, form a partition of Z_i. By the same reasoning as the one for Claim 5.16, we can conclude that $v \in \widetilde{H}(Z_i)$.

In the next claim, we show that T traverses all half-cuts of C, except one.

Claim 5.18. *There is exactly one index $g \in \{1, \ldots, \ell\}$ for which $T \cap \partial Z_{i-1} \cap \partial Z_i = \emptyset$.*

That there is *at least* one such index g, follows from Claim 5.16: If T had an edge in each half-K-cut of C then the edges in all half-K-cuts would form a cycle. Since T is a spanning tree of the subgraph induced by W^C, and since the half-K-cuts represented by C contain all edges connecting different sets Z_i, the uniqueness of g follows.

It remains to prove that the edges of T in the half-K-cuts represented by C form either a back spine satisfying condition (i), or a front spine satisfying condition (ii), or can be divided into two spines that satisfy condition (iii). With Claims 5.16, 5.17 and 5.18, this is just a matter of verifying that these conditions hold.

By Claim 5.16, for each $i \in \{1, \ldots, \ell\} \setminus \{g-1, g\}$, all edges in $T \cap \partial Z_i$ have a common endpoint $w_i \in Z_i$, and $w_i \in \widetilde{H}(Z_i)$ for $i \neq \ell$. Furthermore, if $g > 1$ then w_{g-2} is a common endpoint of $T \cap \partial Z_{g-1} \subseteq T \cap \partial Z_{g-2} \cap \partial Z_{g-1}$, and Claim 5.17 implies that $w_{g-2} \in \widetilde{H}(Z_{g-1})$. Similarly, if $g < \ell$ then $w_{g+1} \in \widetilde{H}(Z_g)$. Therefore:

- If $g = 1$, then $w_2, \ldots, w_{\ell-1}, w_\ell$ is a back spine and $w = w_\ell$ satisfies (i).
- If $g = \ell$, then $w_0, w_1, \ldots, w_{\ell-2}$ is a front spine and $w = w_0$ satisfies (ii).
- If $g \in \{2, \ldots, \ell-1\}$ then $w = w_0, w_1, \ldots, w_{g-2}$ is a front spine, $w_{g+1}, \ldots, w_{\ell-1}, w_\ell = w$ is a back spine, and w satisfies (iii).

This completes the proof of (iv) for the case when $w \in Z_0$.

The other case is when $w \in Z_1 \cup Z_{\ell-1}$, that is w is an in-hub for W^C. Recall that T denotes the witness tree for w and W^C. By the definition of hubs and Z_0, w has a neighbor $w_0 \in Z_0$. By Observation 5.6, $w_0 \in \widetilde{H}(W^C)$ (that is, w_0 is an out-hub for W^C) and $(T \cup \{(w_0, w)\}, w_0)$ is its witness tree for w_0 and W^C. The proof above for the case of out-hubs now implies that w_0 satisfies either condition (i) or (ii); in the first case w satisfies (i) as $w_{\ell-1}$, and in the second case w satisfies (ii) as w_1. □

5.2 The Algorithm

As explained at the beginning of this section, to determine whether $\mathrm{stc}(G) = K$ it is sufficient to compute the hub sets for the basic K-cuts of G. This is because $\mathrm{stc}(G) = K$ if and only if r (the root vertex of G, that has degree K) is a hub for its basic K-cut ∂Z, where $Z = V \setminus \{r\}$.

The algorithm follows a dynamic programming paradigm, processing all basic K-cuts bottom-up along their tree structure, as defined earlier in this section. As presented (see the pseudo-code in Algorithm 1), it only solves the decision version, determining whether $\mathrm{stc}(G) = K$ or not. If $\mathrm{stc}(G) = K$, a spanning tree of G with congestion K can be reconstructed by standard backtracking.

For each basic K-cut ∂Z, the algorithm constructs a set $H(Z)$ intended to contain exactly the hubs for Z. These sets are computed using a recurrence relation established in Section 5.1. We start with external nodes. If $Z = W^a$, for an external node $a = \phi(w)$ of $\mathfrak{C}_G$, then, according to Lemma 5.7, in $H(Z)$ we include w and its neighbors (line 7). If $Z = W^a$ for an internal node a, then $H(Z)$ is computed from the hub sets of its children, using either the recurrence from Lemma 5.9, if a is of Type 1 (lines 10-11), or the recurrence from Lemma 5.13, if a is of Type 0 (lines 14-15). In both cases, this computation can be implemented efficiently using standard data structures.

Algorithm 1 The main algorithm

1: **Input:** Graph $G = (V, E)$ and its a cactus representation $\mathfrak{C}_G, \phi$
2: **if** there exists a node b of $\mathfrak{C}_G$ such that $|\phi^{-1}(b)| > 1$ **then output** NO
3: **choose** a root $r \in V$ of degree K in G
4: **order** the basic K-cuts linearly so that each child precedes its parent
5: **for** each basic K-cut ∂Z, in this ordering **do**
6:     **case** $Z = W^a$ for an external node $a = \phi(w)$ for $w \in V \setminus \{r\}$
7:         $H(W^a) \leftarrow \{w\} \cup N(w)$
8:     **case** $Z = W^a$ for a Type-1 internal node $a = \phi(w)$ for some $w \in V$
9:         **let** $Z_1, \ldots, Z_t$ be the list of all the children of W^a
10:         **if** $w \in H(Z_i)$ for all $i = 1, \ldots, t$ **then** $H(W^a) \leftarrow \{w\} \cup (N(w) \setminus W^a)$
11:         **else** $H(W^a) \leftarrow \emptyset$
12:     **case** $Z = W^a$ for a Type-0 internal node a
13:         **let** $Z_1, \ldots, Z_t$ be the list of all the children of W^a
14:         $\widehat{H} \leftarrow V(\partial W^a) \cap H(Z_1) \cap \cdots \cap H(Z_t)$
15:         $H(W^a) \leftarrow \widehat{H} \cup (N(\widehat{H} \cap W^a) \setminus W^a)$
16:     **case** $Z = W^C$ for a non-trivial cycle C
17:         apply Algorithm 2
18: **if** $r \in H(V \setminus \{r\})$ **then output** YES **else output** NO

The last case, relevant only when K is even, is when $Z = W^C$, for a non-trivial cycle C of $\mathfrak{C}_G$ (the pseudo-code for this case is given separately in Algorithm 2). In this case the algorithm applies the recurrence implicit in Lemma 5.15. In order to do this, for each vertex $w \in Z_1 \cup Z_0 \cup Z_{\ell-1}$ we need to identify back and/or front spines that, based on the cases (i), (ii) or (iii) from this lemma, would imply that w should be added to $H(Z)$.

The challenge is to implement this process efficiently. One key observation here is that (by definition and Observation 5.14), every non-empty suffix of a back spine is also a back spine, and the analogous property applies to front spines. This means that it's sufficient to only compute the maximum spine lengths for each candidate vertex w, not the actual spines. We achieve this using an embedded dynamic programming procedure that processes C in the two directions, and computes $H(Z)$ in time $\tilde{O}(k|C|)$ (the number of edges represented by C).

To give more detail, let's consider the case of back spines (the computation for front spines is symmetric). In this case it is sufficient to calculate, for each candidate vertex $w_i \in Z_i$ of a back spine, the minimum value s for which there exists a path $w_s, \ldots, w_i$ that is a potential prefix of a back spine, i.e., each $w_{i'}$ on this path is an in-hub for $Z_{i'}$ and w_s is also an out-hub for Z_{s-1}. This value s computed by the algorithm is denoted $S^-(w_i)$. It is sufficient to compute $S^-(w_i)$ for the vertices w_i in the candidate set $U_i^- \subseteq Z_i$ which, in accordance with the definition of a back spine and Observation 5.14, contains only in-hubs for Z_i that are also out-hubs for Z_{i-1} (line 5). Two border cases are treated differently: for $i = \ell$, U_ℓ^- is restricted, in a natural way, to out-hubs for $Z_{\ell-1}$ in Z_ℓ and, for $i = 0$, we let $U_1^- = \emptyset$ for technical convenience (line 3). The values $S^-(w_i)$ are then computed by a dynamic program starting from $S^-(w_2) = 2$ for all candidates $w_2 \in Z_2$, and then, for increasing i, setting $S^-(w_i)$ to be the maximum of $S^-(w_{i-1})$ over the neighbors of w_i, or to i if there are no candidate neighbors (line 7).

Once the maximum spine lengths, back and front, are computed, the calculation follows the rules from Lemma 5.15 in a straightforward way (lines 13-16).

Algorithm 2 The subroutine for cycles

Input: Cycle C in $\mathfrak{C}_G$ of length $\ell \geq 3$
let $Z_1, \ldots, Z_{\ell-1}$ be the children of W^C ordered along the cycle C
▷ *computing the back spines*
$U_1^- \leftarrow \emptyset$; $U_\ell^- \leftarrow H(Z_{\ell-1}) \setminus W^C$
for $i = 2, 3, \ldots, \ell$ **do**
  if $i < \ell$ **then** $U_i^- \leftarrow Z_i \cap H(Z_i) \cap H(Z_{i-1})$
  for all $w \in U_i^-$ **do**
    $S^-(w) \leftarrow \max(\{i\} \cup \{S^-(v) \mid v \in N(w) \cap U_{i-1}^-\})$ ▷ for $i = 2$, $S^-(w) = 2$
▷ *computing the front spines*
$U_{\ell-1}^+ \leftarrow \emptyset$; $U_0^+ \leftarrow H(Z_1) \setminus W^C$
for $i = \ell-2, \ell-3, \ldots, 1, 0$ **do**
  if $i \geq 1$ **then** $U_i^+ \leftarrow Z_i \cap H(Z_i) \cap H(Z_{i+1})$
  for all $w \in U_i^+$ **do**
    $S^+(w) \leftarrow \max(\{i\} \cup \{S^+(v) \mid v \in N(w) \cap U_{i+1}^+\})$
▷ *computing the out-hubs in both front and back spines*
$H^0 \leftarrow \{w \in U_0^+ \cap U_\ell^- \mid S^+(w) + 3 \geq S^-(w)\}$
▷ *computing the in-hubs*
$H^- \leftarrow \{w \in U_{\ell-1}^- \cap V(\partial W^C) \mid S^-(w) = 2\}$
$H^+ \leftarrow \{w \in U_1^+ \cap V(\partial W^C) \mid S^+(w) = \ell - 2\}$
▷ *computing the resulting set of hubs*
$H(W^C) \leftarrow H^0 \cup H^- \cup (N(H^-) \setminus W^C) \cup H^+ \cup (N(H^+) \setminus W^C)$

Correctness proof. To prove the correctness of the algorithm, we need to show that it correctly decides whether $\mathrm{stc}(G) = K$. This follows directly from the claim below.

Claim 5.19. *Algorithm 1 (with the subroutine in Algorithm 2) computes the correct hub sets, that is, for each basic K-cut ∂Z we have $H(Z) = \widetilde{H}(Z)$.*

To prove Claim 5.19, we prove that $H(Z) = \widetilde{H}(Z)$ for all basic K-cuts ∂Z inductively, in the order in which the sets $H(Z)$ are calculated by the algorithm.

For basic K-cuts $Z = \partial W^b$, where $b \neq \phi(r)$, the calculation of $H(W^b)$ in Algorithm 1 exactly follows the statements for $\widetilde{H}(W^b)$ in Lemmas 5.7, 5.9, and 5.13 for external nodes, Type-1 internal nodes, and Type-0 internal nodes, respectively. So the inductive claim follows.

For a nontrivial cycle C, as already explained earlier, in the first part of Algorithm 2 we calculate, for each $w = w_\ell \in H(Z_{\ell-1}) \setminus W^C$, the value $S^-(w)$ equal to the minimal s such that a back spine $w_s, \ldots, w_\ell$ exists. Similarly, for each $w = w_0 \in H(Z_1) \setminus W^C$, the value $S^+(w)$ is the maximal s' for which a front spine $w_0, \ldots, w_{s'}$ exists.

It remains to explain the meaning and the computation of sets H^0, H^- and H^+. These are simply the hub sets corresponding to the three different cases in Lemma 5.15.

Consider the computation of H^0. If for some $w = w_0 = w_\ell$ we have $s' + 3 \geq s$ then we select a back spine $w_s, \ldots, w_\ell$ and a front spine $w_0, \ldots, w_{s'}$ that are guaranteed to exist by the previous paragraph. We let $g = s - 1$ and observe that $s' \geq g - 2$ and $g \in \{2, \ldots, \ell - 1\}$. Now $w_0, \ldots, w_{g-2}$ is a front spine, as it is a prefix of the front spine $w_0, \ldots, w_{s'}$ above and $w_{g+1}, \ldots, w_\ell$ is a back spine equal to the back spine above. Thus $w \in \widetilde{H}(W^C)$ by Lemma 5.15(iii). Algorithm 2 calculates H^0 as the set of precisely all these hubs w.

Consider the computation of H^-. (The case of H^+ is symmetric.) Each $w = w_{\ell-1} \in H^-$ is in $V(\partial W^C)$, thus it has a neighbor $w_\ell \in Z_\ell$. We also have $S^-(w_{\ell-1}) = 2$, which now guarantees the existence of a back spine $w_2, \ldots, w_{\ell-1}, w_\ell$. It follows that $w_{\ell-1} \in \widetilde{H}(W^C)$ by Lemma 5.15(i). Thus Algorithm 2 calculates H^- as the set of all in-hubs from Lemma 5.15(i), and $N(H^-) \setminus W^C$ is the set of out-hubs from Lemma 5.15(i).

This completes the proof of the inductive claim for cycles, namely that $H(W^C) = \widetilde{H}(W^C)$ for each non-trivial cycle C. The proof of Claim 5.19 is now complete.

Running time. We now analyze the running time. Recall that $n = |V|$ is the number of vertices of G and $m = |E|$ is the number of edges. As the input graph is K-edge-connected, each vertex has degree at least K, so $m = \Omega(Kn)$. One key property behind our estimate of the running time is that for each basic cut ∂Z, its corresponding hub set $H(Z)$ satisfies $|H(Z)| \leq 2K$. This follows directly from the fact that $H(Z) \subseteq V(\partial Z)$, and $|\partial Z| = K$. Furthermore, the set differences that occur in Algorithms 1 and 2 are also subsets of $V(\partial Z)$, so they can be computed in $\tilde{O}(K)$ time.

We first analyze Algorithm 2. As explained above, all the sets U_i^-, U_i^+, H^0, H^-, H^+ have size $O(K)$. Thus the computation of spines and of the sets H^0, H^-, H^+ involves $O(K\ell)$ operations on integers in $\{1, ..., n\}$ and vertex identifiers. The subsequent computation consists of a constant number of operations on sets of vertices of size K. Since the total length of the cycles in $\mathfrak{C}_G$ is $O(n)$, the overall time for all invocations of Algorithm 2 is $\tilde{O}(Kn) = \tilde{O}(m)$.

Turning to Algorithm 1, the initialization part runs in time $\tilde{O}(m)$, including the construction of the cactus representation $\mathfrak{C}_G$, ϕ (see [12]). Within the dynamic programming process, for each node a of $\mathfrak{C}_G$ of degree d, processing a basic K-cut $\partial Z = \partial W^a$ involves $O(d)$ set operations, each on sets of vertices of size $O(K)$. Since the number of links in $\mathfrak{C}_G$ is $O(n)$, the total time for processing all these K-cuts is $\tilde{O}(Kn) = \tilde{O}(m)$.

Combining the bounds for Algorithms 1 and 2, we conclude that the total running time of our algorithm is $\tilde{O}(m)$. Together with Claim 5.19 this completes the proof of Theorem 1.2.

References

[1] S. Bhatt, F. Chung, T. Leighton, and A. Rosenberg. Optimal simulations of tree machines. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science (FOCS)*, page 274–282, USA, 1986.

[2] H. Bodlaender, F. Fomin, P. Golovach, Y. Otachi, and E. Leeuwen. Parameterized complexity of the spanning tree congestion problem. *Algorithmica*, 64:1–27, 09 2012.

[3] H. L. Bodlaender, K. Kozawa, T. Matsushima, and Y. Otachi. Spanning tree congestion of k-outerplanar graphs. *Discrete Mathematics*, 311(12):1040–1045, 2011.

[4] L. Cai and D. G. Corneil. Tree spanners. *SIAM Journal on Discrete Mathematics*, 8(3):359–387, 1995.

[5] L. S. Chandran, Y. K. Cheung, and D. Issac. Spanning Tree Congestion and Computation of Generalized Györi-Lovász Partition. In *45th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 32:1–32:14, 2018.

[6] E. A. Dinic, A. V. Karzanov, and M. V. Lomonosov. The structure of a system of minimal edge cuts of a graph. In *Issledovani po diskretnoi optimizacii (Studies in discrete optimization)*, pages 290—306. Nauka, Moscow, 1976. In Russian.

[7] F. F. Dragan, F. V. Fomin, and P. A. Golovach. Spanners in sparse graphs. *Journal of Computer and System Sciences*, 77(6):1108–1119, 2011.

[8] S. P. Fekete and J. Kremer. Tree spanners in planar graphs. *Discrete Applied Mathematics*, 108(1):85–103, 2001.

[9] T. Fleiner and A. Frank. A quick proof for the cactus representation of mincuts. Technical Report QP-2009-03, Egerváry Research Group, Budapest, 2009.

[10] S. W. Hruska. On tree congestion of graphs. *Discrete Mathematics*, 308(10):1801–1809, 2008.

[11] K. K., Y. Y., K. S., and Y. M. Spanning tree congestion problem on graphs of small diameter. *RIMS Kokyuroku*, 1941:17–21, 2015.

[12] D. R. Karger and D. Panigrahi. A near-linear time algorithm for constructing a cactus representation of minimum cuts. In *Proc. Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 246–255. SIAM, 2009.

[13] S. Khuller, B. Raghavachari, and N. Young. Designing multi-commodity flow trees. *Information Processing Letters*, 50(1):49–55, 1994.

[14] P. Kolman. Approximating spanning tree congestion on graphs with polylog degree. In *Proc. Int. Workshop on Combinatorial Algorithms (IWOCA)*, pages 497–508, 2024.

[15] P. Kolman. Approximation of spanning tree congestion using hereditary bisection. In *Proc. 42nd Int. Symposium on Theoretical Aspects of Computer Science, STACS*, pages 63:1–63:6, 2025.

[16] K. Kozawa and Y. Otachi. Spanning tree congestion of rook's graphs. *Discussiones Mathematicae Graph Theory*, 31(4):753–761, 2011.

[17] K. Kozawa, Y. Otachi, and K. Yamazaki. On spanning tree congestion of graphs. *Discrete Mathematics*, 309(13):4215–4224, 2009.

[18] M. Lampis, V. Mitsou, E. Nemery, Y. Otachi, M. Vasilakis, and D. Vaz. Parameterized spanning tree congestion. In *50th Int.l Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 65:1–65:20, 2025.

[19] H. Law, S. Leung, and M. Ostrovskii. Spanning tree congestion of planar graphs. *Involve, a Journal of Mathematics*, 7(2):205–226, 2013.

[20] H.-F. Law and M. Ostrovskii. Spanning tree congestion: duality and isoperimetry; with an application to multipartite graphs. *Graph Theory Notes NY*, 58:18–26, 2010.

[21] L. Lin and Y. Lin. The spanning tree congestion problem on interval graphs. *Discrete Applied Mathematics*, 377:147–153, 2025.

[22] C. Löwenstein. *In the Complement of a Dominating Set*. PhD thesis, Technische Universität at Ilmenau, 2010.

[23] H. Luu and M. Chrobak. Better hardness results for the minimum spanning tree congestion problem. *Algorithmica*, 87(1):148–165, 2025.

[24] Y. Okamoto, Y. Otachi, R. Uehara, and T. Uno. Hardness results and an exact exponential algorithm for the spanning tree congestion problem. In *Proc. 8th Annual Conference on Theory and Applications of Models of Computation (TAMC)*, page 452–462, 2011.

[25] M. Ostrovskii. Minimal congestion trees. *Discrete Mathematics*, 285(1):219–226, 2004.

[26] Y. Otachi. *A Survey on Spanning Tree Congestion*, pages 165–172. Springer International Publishing, Cham, 2020.

[27] Y. Otachi, H. L. Bodlaender, and E. J. Van Leeuwen. Complexity results for the spanning tree congestion problem. In *Proc. 36th Int. Conference on Graph-Theoretic Concepts in Computer Science (WG)*, page 3–14, 2010.

[28] A. L. Rosenberg. Graph embeddings 1988: Recent breakthroughs, new directions. In J. H. Reif, editor, *VLSI Algorithms and Architectures*, pages 160–169, New York, NY, 1988. Springer New York.

[29] P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

[30] S. Simonson. A variation on the min cut linear arrangement problem. *Mathematical Systems Theory*, 20:235–252, 12 1987.

A $\mathbb{NP}$-Hardness for Degree 4

In this section, as a warm-up for our $\mathbb{NP}$-completeness proof of problem STC for graphs of degree 3, we show a simpler proof for graphs of degree at most 4. That is, we prove the following theorem.

Theorem A.1. *Problem STC is $\mathbb{NP}$-complete for graphs of degree at most 4.*

The proof is by reduction from problem (M2P1N)-SAT, the version of SAT defined in Section 3 (see also [23]). We show how, given a boolean expression ϕ that is an instance of (M2P1N)-SAT, to compute in polynomial time a graph G and a constant K such that

($*$) ϕ is satisfiable if and only if $\mathrm{stc}(G) \le K$.

Our construction builds on the ideas from [23, 18]; in particular we use the concept of a graph with double-weighted edges, with weights $a\!:\!b$ that are polynomial in the size of ϕ. This is permitted by Lemma B.1 and our construction of $\mathcal{W}(a,b)$ in Appendix B. (Alternatively, for this proof we can use the double-weight gadget of degree 4 constructed in [18].) For brevity, from now on we will refer to double weights simply as "weights". Similarly, the "degree" of a vertex refers to its weighted degree.

Let ϕ be the given instance of (M2P1N)-SAT with n variables and m clauses, of which m' clauses are 2N- or 2P-clauses. We take $K = 3m + 5$, and we convert ϕ into a graph G as follows (see Figure 8):

- For each variable x, create a vertex x and for each clause κ create a vertex κ.
- For each vertex v that is a variable, 2N-clause or 2P-clause, create three other corresponding vertices: *root vertices* r_v^1, r_v^2 and a *terminal vertex* t_v, connected by three unweighted edges (r_v^1, r_v^2), (r_v^1, t_v), and (r_v^2, t_v).
- Add $n + m'$ weight-2 edges arbitrarily so that the root vertices form a cycle called the *root cycle*. The edges in the cycle will be called *root-cycle edges*, and the edges connecting the root cycle to terminal vertices are *root-terminal edges*.
- For each variable x, add a *root-variable* edge (x, t_x) with weight $1\!:\!K-5$.
- For each 2P-clause γ, add an edge (γ, t_γ) with weight $1\!:\!K-1$, and for each 2N-clause α, add an edge (α, t_α) with weight $2\!:\!K-1$. Call these edges *root-clause* edges.
- For each clause κ, add an edge from κ to each vertex representing a variable whose literal (positive or negative) appears in κ. If κ is a positive clause, these edges have weight $1\!:\!K-2$, and if κ is a negative clause, these edges have weight $1\!:\!K-3$. Call such edges *clause-variable* edges.

In the proof, when discussing G, for brevity, we will refer to the vertex representing a variable x simply as "variable x" and, similarly, to the vertex representing a clause κ as "clause κ".

It now remains to show that G and K satisfy condition ($*$). We prove the two implications in ($*$) separately.

($\Rightarrow$) Given a truth assignment that satisfies ϕ, we can construct a spanning tree T for G by adding to it the following edges (see Figure 9):

- each root-variable edge,
- for each clause κ, exactly one clause-variable edge from κ to any variable whose literal satisfies κ (if κ is satisfied by multiple literals, choose this variable arbitrarily),
- all edges in the root cycle except for one edge of weight 2,

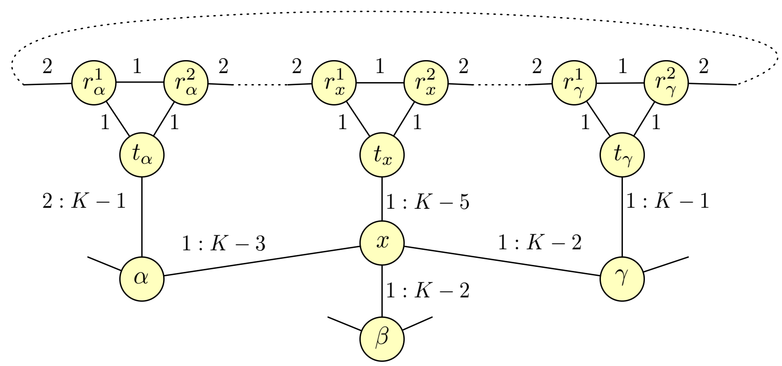

Figure 8: The construction of G.

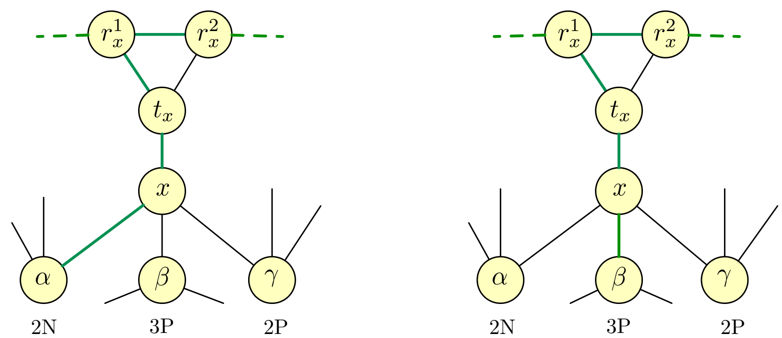

Figure 9: Converting a truth assignment satisfying ϕ into a spanning tree T. The picture shows a variable x with its 2N-clause α, 3P-clause β and 2P-clause γ. Tree edges are shown using thick (green) lines. On the left, the case when x is false and edge (x, α) is chosen for α. On the right, the case when x is true and edge (x, β) is chosen for β. Three other cases are not shown: when x is chosen for γ, when x is chosen for both β and γ, and when x is not chosen at all.

- for each variable or clause u, edge (t_u, r^1_u).

By inspection, T forms a spanning tree, and each clause is a leaf in T. Furthermore, for each variable x, if x has an edge in T to its 2N-clause then x does not have an edge in T to any of its positive clauses. To complete the proof of the ($\Rightarrow$) implication, we show that each edge e in T has congestion at most K. For this, we consider several cases.

Case 1: e is in the root cycle or it is a root-terminal edge. Consider first the sub-case when $e = (r^1_v, r^2_v)$, for some v (variable or clause). Then e contributes 1 to its induced cut, (r^2_v, t_v) contributes 1, and the only edge from the root cycle not contained in T contributes 2. All other edges in this cut are clause-variable and root-clause edges and there are at most $3m$ of them (because clauses are leaves in T). Thus, $\text{cng}_T(e) \leq 4 + 3m < K$.

The second sub-case is when $e = (r^2_u, r^1_v)$, for some $u \neq v$ (variables or clauses), is very similar. Edge e contributes 2 to its cut, the only edge from the root cycle not contained in T contributes 2, and the clause-variable and root-clause edges contribute $3m$ at most, so $\text{cng}_T(e) \leq 4 + 3m < K$ as well.

In the third sub-case when e is a root-terminal edge (t_v, r^1_v), for some v, e contributes 1 to its cut, and (t_v, r^2_v) contributes 1. If v is a clause, then the only other edge in the cut is a root-clause edge of weight at most 2; if v is a variable, then all other edges in the cut are at most 5 of the 9 clause-variable edges from the clauses of x. Thus, $\text{cng}_T(e) \leq 2 + 5 = 7 < K$.

Case 2: e is a root-variable edge $e = (x, t_x)$ for some variable x. Let α, β, γ be the 2N-clause, 3P-clause, and 2P-clause of x, respectively. The shore T_e^{+x} of x of the cut induced by e contains x and can also contain some of its clauses, but if it contains α then it contains none of β, γ. So T_e^{+x} is either (i) $\{x\}$, or (ii) $\{x, \kappa\}$ for $\kappa \in \{\alpha, \beta, \gamma\}$, or (iii) $\{x, \beta, \gamma\}$. In case (i), $\text{cng}_{G,T}(e) = (K-5)+3 \leq K-2$. In case (ii), $\text{cng}_{G,T}(e) \leq (K-5) + 2 + 3 = K$. (It is equal K only when $\kappa = \alpha$.) In case (iii), $\text{cng}_{G,T}(e) = (K-5) + 1 + 2 + 2 = K$.

Case 3: e is a clause-variable edge $e = (x, \kappa)$, for some clause κ of x. If $\kappa = \alpha$ then $\text{cng}_{G,T}(e) = (K-3) + 3 = K$, and if $\kappa \in \{\beta, \gamma\}$ then $\text{cng}_{G,T}(e) = (K-2) + 2 = K$.

($\Leftarrow$) Let T be a spanning tree with $\text{cng}_G(T) \leq K$. We show how to convert T into a satisfying assignment for ϕ. It is sufficient to prove the following lemma:

Lemma A.2. *Tree T has the following properties:*

(a) T does not contain any root-clause edges.

(b) If a variable x is connected in T to its 2N-clause, then x is not connected in T to its 2P-clause or its 3P-clause.

The reason this lemma is sufficient is because it allows us to produce a satisfying assignment for ϕ: For each variable x, if x is connected to its 2N-clause in T, make x false; otherwise make it true. By Lemma A.2(b), this is a valid truth assignment, and by Lemma A.2(a), every clause vertex κ has an edge in T to some of its variables, so the definition of the truth assignment guarantees that this variable will satisfy κ.

Proof. The proof of Lemma A.2(a) is straightforward: Suppose that a 2N- or 2P-clause κ has its root-clause edge $f = (\kappa, t_\kappa)$ in T. Let x, y be the two variables of the literals in κ. Then there are two disjoint paths from κ to t_κ in G that do not use f: namely paths starting with $\kappa - x - t_x$ and $\kappa - y - t_y$, and then following the root cycle to t_κ, for one path clockwise and for the other one counter-clockwise, to ensure that the paths are disjoint. The cut ∂T_f induced by f must contain at least one edge from each of these two paths, implying $\text{cng}_T(f) \geq K-1+2 = K+1$, a contradiction.

The rest of this section is devoted to the proof of Lemma A.2(b), which will complete the proof of $\mathbb{NP}$-completeness. We start with the following claim:

Claim A.3. *For any two root vertices, the path in T between these vertices consists of only root-cycle edges and root-terminal edges.*

To justify Claim A.3, consider any two different root vertices p and q and suppose that the p-to-q path P in T does not satisfy Claim A.3. Since, by Lemma A.2(a), P does not contain any root-clause edges, P must contain at least one clause-variable edge, say $f = (y, \kappa)$ (cf. Figure 10). But then in the cut ∂T_f induced by f, the vertices p and q would be in different shores, so this cut would have to contain two root-cycle edges, or a root-cycle edge and two root-terminal edges, or four root-terminal edges; in any case, the total weight of these edges is 4. Thus, $\text{cng}_T(f) \geq K - 3 + 4 = K + 1$ – contradicting the assumption that $\text{cng}_G(T) \leq K$. So Claim A.3 holds.

Continuing with the proof of Lemma A.2(b), let x be any variable, and let α, β, γ be its 2N-clause, 3P-clause, and 2P-clause. Assume for contradiction that edges (α, x) and (π, x) are in T for some $\pi \in \{\beta, \gamma\}$. Consider the path P in T connecting $\{\alpha, x, \pi\}$ to the root cycle. By Lemma A.2(a), P must use a root-variable edge, and let $e = (y, t_y)$ be the first such edge. We will examine the edges crossing the cut ∂T_e induced by e. From Claim A.3, we obtain:

Corollary A.4. *The root cycle is on the shore $T_e^{+t_y}$ of ∂T_e, while α, x, π are all on the shore T_e^{+y}.*

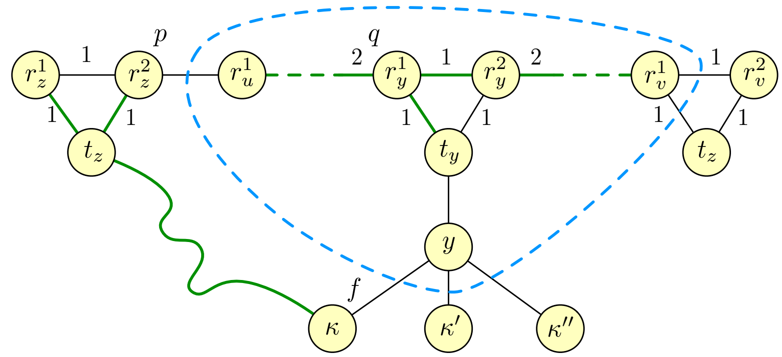

Figure 10: The cut of a clause-variable edge $f = (\kappa, y)$ that is on a path between two different root vertices p and q, implying $\mathrm{cng}_T(f) \geq K + 1$. Tree edges are shown using thick (green) lines. The cut is indicated using a (blue) dashed curve.

By the definition of (M2P1N)-SAT, different clauses in ϕ cannot share more than one variable. This implies that there are four distinct variables z_1, z_2, z_3, z_4, none equal to x, with clause-variable edges from α, β, γ, namely such that $z_1 \in \alpha$, $z_2, z_3 \in \beta$, and $z_4 \in \gamma$. This in turn implies that there are 7 edge-disjoint paths in G from the set $\{\alpha, x, \pi\}$ to the root cycle. These paths are:

- $\alpha - t_\alpha - r^1_\alpha$, $\alpha - t_\alpha - r^2_\alpha$ (these count as disjoint because $w_1(\alpha, t_\alpha) = 2$), $\alpha - z_1 - t_{z_1} - r^1_{z_1}$,
- $x - t_x - r^1_x$,
- if $\pi = \beta$: $\beta - z_2 - t_{z_2} - r^1_{z_2}$, $\beta - z_3 - t_{z_3} - r^1_{z_3}$, $x - \gamma - t_\gamma - r^1_\gamma$.
- if $\pi = \gamma$: $\gamma - t_\gamma - r^1_\gamma$, $\gamma - z_4 - t_{z_4} - r^1_{z_4}$, $x - \beta - z_2 - t_{z_2} - r^1_{z_2}$.

All these 7 paths cross cut ∂T_e, and at most one of them can use edge e to cross it. So the congestion of e is at least $\mathrm{cng}_T(e) \geq K - 5 + 6 = K + 1$, contradicting the choice of T, and completing the proof of Lemma A.2. □

B The Double-Weight Gadget

In this section we describe the construction of our double-weight gadget $\mathcal{W}(a, b)$, that will be used to replace edges with double weights in our $\mathbb{NP}$-completeness proof in Section 3. The double-weight gadget is constructed from a sub-gadget that we call a *bottleneck*, that we introduce first.

The bottleneck gadget. For an integer $w \geq 3$, we define a *w-bottleneck of degree* 3 as an unweighted graph $\mathcal{B} = \mathcal{B}(w)$ with two distinguished vertices s, t called the *gates of* $\mathcal{B}$, that has the following properties:
(b1) $\mathrm{stc}(\mathcal{B}) = w$,
(b2) for any spanning tree T of $\mathcal{B}$, the s-to-t path in T contains an edge e with $\mathrm{cng}_{\mathcal{B},T}(e) \geq w$,
(b3) the degrees of s and t are 2 and all other degrees in $\mathcal{B}$ are at most 3.

We now show a construction of $\mathcal{B}(w)$. We remark that congestion properties of some grid-like graphs of degree 3 have been analyzed in the literature, for example in [19] for hexagonal grids. However, these results are not sufficient for our purpose because they don't explicitly address condition (b2) of bottleneck graphs.

Let $w \geq 3$ be the specified parameter. Our gadget is a hexagonal grid, depicted as a slanted wall-of-bricks grid of dimensions $w \times w$, illustrated in Figure 11 for $w = 4$ and $w = 5$. More precisely, geometrically the grid has $w - 1$ rows of 2×1 rectangular faces called *bricks*, with $w - 1$ bricks per row. The vertices are the corners of the rectangles, with the edges represented by lines connecting these corners. The gate s of $\mathcal{B}$ is the bottom left corner while t is chosen to be the top right corner. The gates have degree 2 and all other vertices have degree 2 or 3, so property (b3) is satisfied.

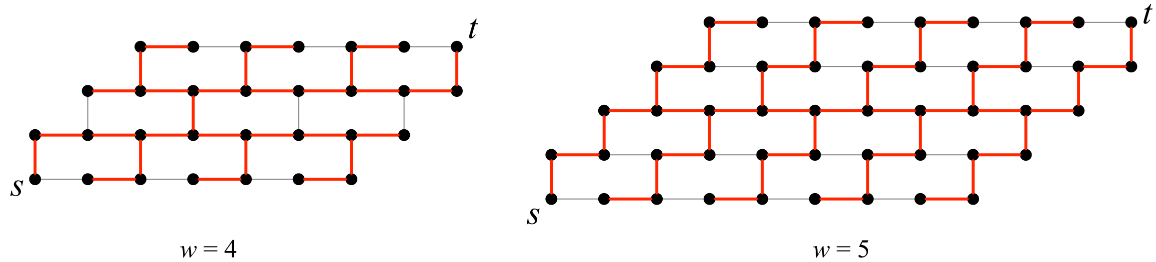

Figure 11: Bottleneck gadget $\mathcal{B}$ for $w = 4$ and $w = 5$. Their spanning trees $\widetilde{T}$ with congestion w are shown in thick (red).

First, we claim that $\mathcal{B}$ has property (b1) that $\mathrm{stc}(\mathcal{B}) = w$. It is sufficient to prove that $\mathrm{stc}(\mathcal{B}) \leq w$, because the other inequality follows from (b2). To this end, we specify a spanning tree $\widetilde{T}$ of $\mathcal{B}$ with congestion w (See Figure 11.) For odd w, $\widetilde{T}$ includes the middle horizontal path, with vertical branches alternatively connecting to the top and bottom boundaries. For even w, $\widetilde{T}$ includes both middle horizontal paths, with one of the two middle vertical edges connecting them. The vertical branches are analogous to the odd case. By inspection, the congestion of $\widetilde{T}$ in both cases is w. Specifically, for odd w, excluding the edges on the left and right boundaries, the edges in the middle horizontal path and the vertical edges touching this path have congestion w. All other edges have lower congestion. For even w, the connecting edge in the middle row has congestion w, and the edges along the middle horizontal paths touching the central brick also have congestion w. All other edges have lower congestion. Note that in both cases, the s-to-t path in $\widetilde{T}$ includes some of congestion-w edges.

Next, we prove property (b2), namely that *for any spanning tree* T of $\mathcal{B}$, at least one of the edges in the s-to-t path P in T has congestion at least w. To do this, it is convenient to express the argument in terms of the dual graph. This follows the approach in [10, 20], where the authors studied congestion in rectangular and triangular grid graphs. Let $\mathcal{B}_{st} = \mathcal{B} + (s,t)$, a graph obtained from $\mathcal{B}$ by adding an edge (s,t), that splits the outer face of $\mathcal{B}$ into two faces. T remains a valid spanning tree of $\mathcal{B}_{st}$. Let $\hat{\mathcal{B}}_{st}$ be the dual graph of $\mathcal{B}_{st}$. ($\hat{\mathcal{B}}_{st}$ has some parallel edges, because $\mathcal{B}$ has some vertices of degree 2.) Define $z_{i,j}$ to be the dual vertex in $\hat{\mathcal{B}}_{st}$ representing the brick at (skewed) column i and row j, where $1 \leq i, j \leq w-1$. We also define two special dual vertices: $z_{w,0}$ is the dual vertex of the outer face whose boundary includes the bottom and right boundaries of $\mathcal{B}$, and $z_{0,w}$ corresponds to the outer face whose boundary include the top and left boundaries of $\mathcal{B}$ (See Figure 12.)

Let $\hat{T}$ be the graph obtained from $\hat{\mathcal{B}}_{st}$ by removing the edges dual to the edges of T. Then $\hat{T}$ is connected and acyclic, so it is a spanning tree of $\hat{\mathcal{B}}_{st}$, and is referred to as the tree dual to T.

Now consider the dual vertices corresponding to the bricks on the main diagonal of $\mathcal{B}$ namely the vertices $z_{i,j}$ where $i + j = w$. Let C be the set of (horizontal) edges of $\mathcal{B}$ dual to the edges of the path $z_{0,w} - z_{1,w-1} - z_{2,w-2} - \cdots - z_{w-1,1} - z_{w,0}$ in $\hat{\mathcal{B}}_{st}$. C is a cut of $\mathcal{B}$ (consisting of the edges along the main diagonal), and therefore P contains at least one edge from C, say an edge $e \in T$ whose dual is $\hat{e} = (z_{p,q}, z_{p+1,q-1})$. Note $z_{p,q}$ is above e while $z_{p+1,q-1}$ is below it. Since the cycle $P + (s,t)$ encloses the top-left region of $\mathcal{B}_{st}$ (more precisely, the corresponding dual edges form a cut of $\mathcal{B}$ that separates $z_{0,w}$ from $z_{w,0}$), there are two disjoint paths Q, Q' in $\hat{T}$ that start

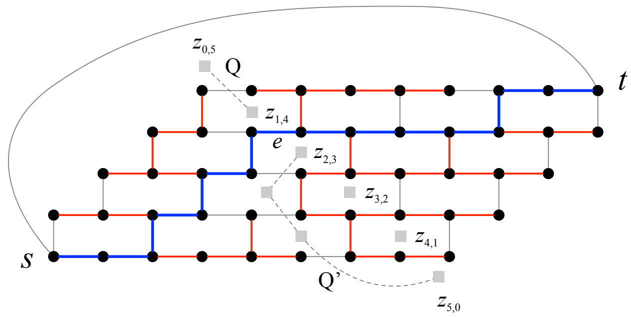

Figure 12: Illustration of an s-t path P in thick (blue) edges that includes an edge e with congestion $w = 5$.

at $z_{p,q}$ and $z_{p+1,q-1}$ and end at $z_{0,w}$ and $z_{w,0}$, respectively (See Figure 12 for an example). Let $l(Q), l(Q')$ denote their respective lengths. Both Q, Q' must cross the boundaries of $\mathcal{B}$. If the two boundaries they cross are top and bottom then $l(Q) + l(Q') \geq w - q + (q - 1) = w - 1$; if they are left and right then $l(Q) + l(Q') \geq p + w - (p + 1) = w - 1$. Otherwise, if the two boundaries are top and right, then $l(Q) + l(Q') \geq w - q + w - (p + 1) = w - 1$; if they are left and bottom then $l(Q) + l(Q') \geq p + q - 1 = w - 1$. In all cases, $l(Q) + l(Q') \geq w - 1$. It is not difficult to see that the congestion of e is equal to $l(Q) + l(Q') + 1$; for a detailed proof of this observation see the analyses of rectangular grids in [10, Section 3.3] and triangular grids in [20, pp. 6], that both naturally apply also to our wall-of-bricks gadget. This means $\mathrm{cng}_{G,T}(e) \geq w$ which completes the proof for property (b2).

Double-weight gadget. To construct *a double-weight gadget* $\mathcal{W}(a, b)$ *of degree* 3, where $a : b$ is a double weight with $a < b$, create a disjoint copies $\mathcal{B}_1, ..., \mathcal{B}_a$ of $\mathcal{B}(b - a + 1)$ (constructed above), where the gates of each $\mathcal{B}_i$ are s_i and t_i. Then add two additional vertices s^* and t^* called the *ports of* $\mathcal{W}(a, b)$, with s^* connected by edges to each gate s_i and t^* is connected by edges to each gate t_i. (See Figure 13.)

The lemma below, establishing the validity of converting a bottleneck sub-gadget into a double-weight gadget $\mathcal{W}(a, b)$, was shown in [23] for some specific choice of the bottleneck sub-gadget, but it trivially extends to our more general formulation, so we omit its proof here.

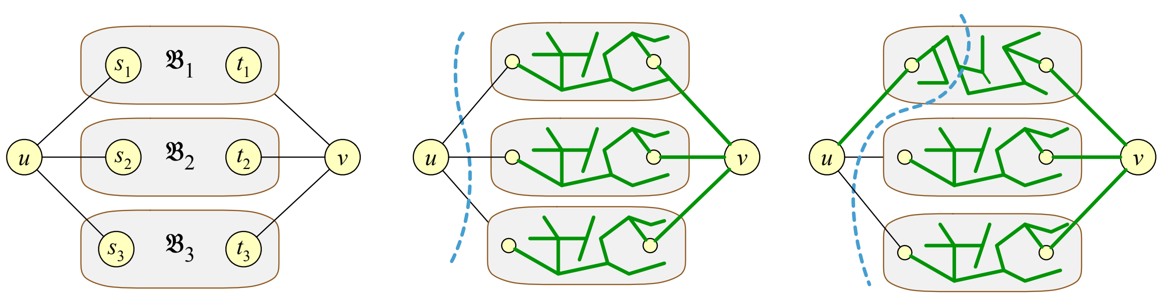

Figure 13: On the left, the double-weight gadget $\mathcal{W}(a, b)$ replacing an edge (u, v) of G with weight $a : b$ to produce a new graph G'. Here, $a = 3$. The pictures in the middle and right illustrate how this gadget "simulates" the double weight. If (u, v) is not in the spanning tree of G, the corresponding spanning tree of G' can traverse the gadget as in the middle picture, contributing a to the congestion. If (u, v) is in the spanning tree of G, the corresponding spanning tree of G' can traverse the gadget as in the picture on the right, contributing $(b - a + 1) + a - 1 = b$ to the congestion.

Lemma B.1. *Let $K \geq 3$ be an integer, G a double-weighted graph of maximum degree $\Delta \geq 3$, and $e = (u, v) \in E$ an edge in G with double weight $a\!:\!b$ satisfying $a < b$ and $b - a \leq K - 2$. Let G' be the graph obtained from G by removing e and replacing it by a degree-3 double-weight gadget $\mathcal{W}(a, b)$ whose ports s^* and t^* are identified with u and v, respectively. Then*

(i) *The (weighted) degrees of the original vertices of G (including u, v) remain unchanged. Thus the maximum degree in G' is Δ.*

(ii) $\mathrm{stc}(G) \leq K$ *if and only if* $\mathrm{stc}(G') \leq K$.