Hardness Results and an Exact Exponential Algorithm for the Spanning Tree Congestion Problem

Yoshio Okamoto[*1], Yota Otachi[*2], Ryuhei Uehara[*3] and Takeaki Uno[*4]

1. Introduction

Spanning tree congestion is a graph parameter defined by Ostrovskii[19] in 2004. Simonson[22] also studied the same parameter under a different name as a variant of cutwidth. After Ostrovskii[19], several graph-theoretic results have been presented[3),6),12)–18),20)], and very recently the complexity of the problem for determining the parameter has been studied[2),21)]. The parameter is defined as follows. Let G be a connected graph and T be a spanning tree of G. The *detour* for an edge $\{u, v\} \in E(G)$ is a unique u–v path in T. We define the *congestion* of $e \in E(T)$, denoted by $cng_{G,T}(e)$, as the number of edges in G whose detours contain e. The *congestion of G in T*, denoted by $cng_G(T)$, is the maximum congestion over all edges in T. The *spanning tree congestion* of G, denoted by $stc(G)$, is the minimum congestion over all spanning trees of G. We denote by STC the problem of determining whether a given graph has spanning tree congestion at most given k. If k is fixed, then we denote the problem by k-STC.

Bodlaender, Fomin, Golovach, Otachi, and van Leeuwen[2),21)] studied the complexity of STC and k-STC. They showed that k-STC is linear-time solvable for apex-minor-free graphs and bounded-degree graphs, while k-STC is NP-complete even for K_6-minor-free graphs with only one vertex of unbounded degree if $k \geq 8$. They also showed that STC is NP-complete for planar graphs. Bodlaender, Kozawa, Matsushima, and Otachi[3)] showed that the spanning tree congestion can be determined in linear time for outerplanar graphs. Although several complexity results are known as mentioned above, they are restricted to sparse graphs. The complexity for non-sparse graphs such as chordal graphs and chordal bipartite graphs were unknown.

In this paper, we show that STC is NP-complete for these important non-sparse graph classes. More precisely, we show that STC is NP-complete even for chain graphs and split graphs. It is known that every chain graph is chordal bipartite, and every split graph is chordal. The hardness for chain graphs is quite unexpected, since there is no other natural graph parameter that is known to be NP-hard for chain graphs, to the best of our knowledge. The hardness for chain graphs also implies the hardness for graphs of clique-width at most three. To cope with the hardness of the problem, we present a fast exponential-time exact algorithm. Our algorithm runs in $O^*(2^n)$ time, while a naive algorithm that examines all spanning trees runs in $O^*(2^m)$ or $O^*(n^n)$ time, where n and m denote the number of vertices and the number of edges. Note that $O^*(f(n)) = O(f(n) \cdot \mathrm{poly}(n))$. The idea, which allows us to achieve this running time, is to enumerate all possible combinations of cuts instead of all spanning trees. Using this idea, we can design a dynamic-programming-based algorithm that runs in $O^*(3^n)$ time. Then, by carefully applying the fast subset convolution method developed by Björklund, Husfeldt, Kaski, and Koivisto[1)], we finally get the running time $O^*(2^n)$. We also study the problem on cographs. It is known that cographs are precisely the graphs of clique-width at most two. For some cographs such as complete graphs and complete p-partite graphs, the closed formulas for the spanning tree congestion are known[12),14),16),19)]. Although the complexity of STC for cographs remains unsettled, we provide a constant-factor approximation algorithm for them. Furthermore, we present a linear-time algorithm for chordal cographs.

Graphs in this paper are finite, simple, and connected, if not explicitly stated otherwise. We deal with edge-weighted graphs in Subsections 1.2 and 2.1. Our exponential-time exact algorithm runs in $O^*(2^n)$ time for edge-weighted graphs, too.

1.1 Graphs

Let G be a connected graph. For $S \subseteq V(G)$, we denote by $G[S]$ the subgraph induced

*1 Center for Graduate Education Initiative, JAIST, Asahidai 1–1, Nomi, Ishikawa 923–1292, Japan. okamotoy@jaist.ac.jp
*2 Graduate School of Information Sciences, Tohoku University. Sendai 980–8579, Japan. JSPS Research Fellow. otachi@dais.is.tohoku.ac.jp
*3 School of Information Science, JAIST, Asahidai 1–1, Nomi, Ishikawa 923–1292, Japan. uehara@jaist.ac.jp
*4 National Institute of Informatics, 2–1–2 Hitotsubashi, Chiyoda-ku, Tokyo, 101–8430, Japan. uno@nii.ac.jp

Vol.2011-AL-136 No.5
2011/9/6

by S. For an edge $e \in E(G)$, we denote by $G - e$ the graph obtained from G by the deletion of e. Similarly, for a vertex $v \in V(G)$, we denote by $G - v$ the graph obtained from G by the deletion of v and its incident edges. By $N_G(v)$, we denote the (*open*) *neighborhood* of v in G; that is, $N_G(v)$ is the set of vertices adjacent to v in G. For $S \subseteq V(G)$, we denote $\bigcup_{v \in S} N_G(v)$ by $N_G(S)$. We define the *degree* of v in G as $deg_G(v) = |N_G(v)|$. If $deg_G(v) = |V(G)| - 1$, then v is a *universal vertex* of G.

Let G and H be graphs. We say that G and H are *isomorphic*, and denote it by $G \simeq H$, if there is a bijection $f \colon V(G) \to V(H)$ such that $\{u, v\} \in E(G)$ if and only if $\{f(u), f(v)\} \in E(H)$. Now assume $V(G) \cap V(H) = \emptyset$. Then the *disjoint union* of G and H, denoted by $G \cup H$, is the graph with the vertex set $V(G) \cup V(H)$ and the edge set $E(G) \cup E(H)$. The *join* of G and H, denoted by $G \oplus H$, is the graph with the vertex set $V(G) \cup V(H)$ and the edge set $E(G) \cup E(H) \cup \{\{u, v\} \mid u \in V(G), v \in V(H)\}$.

For $A, B \subseteq V(G)$, we define $E_G(A, B) = \{\{u, v\} \in E(G) \mid u \in A,\ v \in B\}$. For $S \subseteq V(G)$, we define the *boundary edges* of S, denoted by $\theta_G(S)$, as $\theta_G(S) = E_G(S, V(G) \setminus S)$. Note that $\theta_G(\emptyset) = \theta_G(V(G)) = \emptyset$. The congestion $cng_{G,T}(e)$ of an edge $e \in E(T)$ satisfies $cng_{G,T}(e) = |\theta_G(A_e)|$, where A_e is the vertex set of one of the two components of $T - e$. For an edge e in a tree T, we say that e *separates* A and B if $A \subseteq A_e$ and $B \subseteq B_e$, where A_e and B_e are the vertex sets of the two components of $T - e$. Clearly, if T is a spanning tree of G and $e \in E(T)$ separates A and B, then $cng_{G,T}(e) \geq |E(A, B)|$. If e separates A and B, we also say that e *divides* $A \cup B$ into A and B.

Let T be a tree rooted at $r \in V(T)$. Then we denote by $prt_T(v)$ the parent of $v \in V(T)$ in T. The parent of the root r is not defined. We denote by $Ch_T(v)$ the children of $v \in V(T)$ in T. Clearly, $N_T(v) = \{prt_T(v)\} \cup Ch_T(v)$ for every non-root vertex v.

1.2 Spanning tree congestion of weighted graphs

A graph G may be associated with an edge-weight function $wei \colon E(G) \to \mathbb{Z}^+$. If a graph has such a function, then we call it an *edge-weighted graph* or just a *weighted graph*. Note that unweighted graphs can be considered as weighted graphs by setting $wei(e) = 1$ for each edge e. For an edge-weighted graph G and $F \subseteq E(G)$, we define $wei(F) = \sum_{f \in F} wei(f)$ for $F \subseteq E(G)$. We extend the notion of spanning tree congestion to edge-weighted graphs by defining the congestion of an edge e as the sum of the weights of edges whose detours pass through the edge e. If $e \in E(T)$ separates vertex sets A and B, then $cng_{G,T}(e) \geq wei(E(A, B))$.

For a weighted graph G, we define the *weighted degree* of v in G as $wdeg_G(v) = wei(\theta_G(\{v\}))$. It is not difficult to see that the following fact holds.

Proposition 1.1. *Let G be a weighted graph, and let $S \subseteq V(G)$. Then*
$$wei(\theta_G(S)) = \sum_{v \in S} wdeg_G(v) - 2wei(E(G[S])).$$

It is known that STC for weighted graphs is equivalent to STC for unweighted graphs in the following sense.

Lemma 1.2 ([2,21]). *Let G be a weighted graph and let $e \in E(G)$. Let G' be the graph obtained from G by removing the edge e and adding $wei(e)$ internally disjoint paths of arbitrary lengths between the ends of e, where each edge in the added paths is of unit weight. Then, $stc(G) = stc(G')$.*

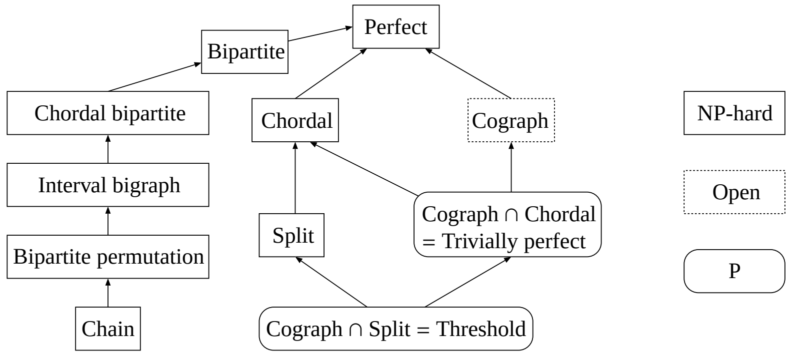

Fig. 1 Relations among graph classes.

1.3 Graph classes

Here we introduce some graph classes. **Fig. 1** depicts relations among graph classes. For graph classes not defined in this subsection see textbooks on graph classes[4),10)].

A graph is *chordal* if it has no induced cycle of length greater than three. A graph G is a *split graph* if its vertex set $V(G)$ can be partitioned into two sets C and I so that C is a clique of G and I is an independent set of G. Clearly, every split graph is a chordal graph (see[10)]). A *cograph* (or *complement-reducible graph*) is a graph that can be constructed recursively by the following rules:

(1) K_1 is a cograph;

(2) if G and H are cographs, then so is $G \cup H$;

(3) if G and H are cographs, then so is $G \oplus H$.

Note that if G is a connected cograph with at least two vertices, then G can be expressed as $G_1 \oplus G_2$ for some nonempty cographs G_1 and G_2. A cograph is a *chordal cograph* if it is also a chordal graph. Chordal cographs are also known as *trivially perfect graphs*[4),10)] and *quasi-threshold graphs*[23)]. It is known that in the construction of a chordal cograph by the above rules, we can assume one of two operands of $\oplus$ is K_1[23)].

Analogous to chordal graphs, *chordal bipartite graphs* are defined as the bipartite graphs without induced cycle of length greater than four. A bipartite graph $G = (X, Y; E)$ is a *chain graph* if there is an ordering $<$ on X such that $u < v$ implies $N_G(u) \subseteq N_G(v)$. It is known that every chain graph is $2K_2$-free[24)], and thus chordal bipartite.

Clique-width is a graph parameter which generalizes treewidth in some sense. Many hard problems can be solved efficiently for graphs of bounded clique-width. It is known that every chain graph has clique-width at most three[5)], and that a graph has clique-width at most two if and only if it is a cograph[7)]. For the definition and further information of clique-width, see a recent survey by Hliněný, Oum, Seese, and Gottlob[11)].

2. Hardness for split graphs and chain graphs

This section presents our hardness results for split graphs and chain graphs. Namely, we prove the following theorems.

Theorem 2.1. *STC is NP-complete for split graphs.*

Theorem 2.2. *STC is NP-complete for chain graphs.*

Since every chain graph has clique-width at most three, we have the following corollary.

Corollary 2.3. *STC is NP-complete for graphs of clique-width at most three.*

The weighted edge argument[2),21)] allows us to present a simple proof for split graphs. However, we are unable to present a simple proof based on the weighted edge argument for chain graphs. This is because, in the process of modifying a weighted graph to an unweighted graph, we may introduce many independent edges (see Lemma 1.2). Although we need somewhat involved arguments for chain graphs, the proofs are based on essentially the same idea.

Clearly, STC is in NP. The proofs of NP-hardness are done by reducing the following well-known NP-complete problem to STC for both graph classes.

Problem: 3-Partition[9)] [SP15]

Instance: A multi-set $A = \{a_1, a_2, \dots, a_{3m}\}$ of $3m$ positive integers and a bound $B \in \mathbb{Z}^+$ such that $\sum_{a_i \in A} a_i = mB$, $a_1 \le a_2 \le \cdots \le a_{3m}$, and $B/4 < a_i < B/2$ for each $a_i \in A$.

Question: Can A be partitioned into m disjoint sets $A_1, A_2, \dots, A_m$ such that, for $1 \le i \le m$, $\sum_{a \in A_i} a = B$? (Thus each A_i must contain exactly three elements from A.)

It is known that 3-Partition is NP-complete in the strong sense[9)]. Thus we assume $a_{3m} \le \mathrm{poly}(m)$, where $\mathrm{poly}(m)$ is some polynomial on m. By scaling each $a \in A$, we can also assume that $a_1 \ge 3m + 2$, $m \ge 3$, $B \ge 8$, and $B/4 + 1 \le a_i \le B/2 - 1$.

2.1 Hardness for split graphs

In this subsection, we prove that STC is NP-hard for split graphs. We first show that STC is NP-hard for edge-weighted split graphs with weighted edges only in the maximum clique, by reducing an instance A of 3-Partition to an edge-weighted split graph G_A such that A is a yes instance if and only if $stc(G_A) \le k$ for some k. We then show that G_A can be modified to an unweighted split graph G'_A in polynomial time so that $stc(G_A) = stc(G'_A)$. This proves Theorem 2.1.

Let A be an instance of 3-Partition. We now construct G_A from A in polynomial time (see **Fig. 2**). Let $I = \{u_i \mid 1 \le i \le 3m\}$ and $C = \{x\} \cup V \cup W$, where $V = \{v_i \mid 1 \le i \le m\}$ and $W = \{w_i \mid m + 1 \le i \le a_{3m}\}$. The graph G_A has vertex set $I \cup C$. The sets I and C are independent set and a clique of G_A, respectively. Each $u_i \in I$ is adjacent to all vertices in V and vertices $w_1, w_2, \dots, w_{a_i}$. More formally, $E(G_A)$ is defined as follows:

$$\begin{aligned} E(G_A) = &\{\{c, c'\} \mid c, c' \in C\} \cup \{\{u, v\} \mid u \in I, v \in V\} \\ &\cup \{\{u_i, w_j\} \mid u_i \in I, m + 1 \le j \le a_i\}. \end{aligned}$$

Recall that $a_i > m$ for any $i \ge 1$. The degrees of vertices in G_A can be determined as follows: $deg_{G_A}(u_i) = a_i$, $deg_{G_A}(v_i) = |C| + |I| - 1$, and $deg_{G_A}(w_i) = |C| + |\{j \mid a_j \ge i\}| - 1$. Some edges of G_A have heavy weights. Let $k = 2B + 2|C| + 2|I| - 15$. Then

$$wei(e) = \begin{cases} \alpha := (k+1)/2 & \text{if } e = \{x, v_i\}, \\ \beta_i := k - deg_{G_A}(w_i) + 1 & \text{if } e = \{x, w_i\}, \\ 1 & \text{otherwise.} \end{cases}$$

Clearly, G_A is a split graph with weighted edges only in the clique C. The weighted degrees of vertices in G_A is as follows: $wdeg_{G_A}(u_i) = a_i$, $wdeg_{G_A}(v_i) = \alpha + |C| + |I| - 2 = k - B + 6$, and $wdeg_{G_A}(w_i) = k$.

Lemma 2.4. *Let $k = 2B + 2|C| + 2|I| - 15$. Then A is a yes instance if and only if $stc(G_A) \le k$.*

Proof. Omitted. □

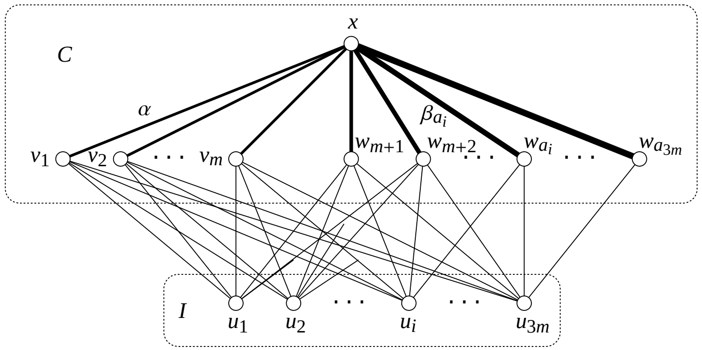

Fig. 2 Reduction. (Unweighted edges in C are not depicted.)

Now we prove the NP-hardness of STC for unweighted split graphs. To this end, we first reduce an instance A of 3-Partition to a weighted split graph G_A as stated above. Recall that all weighted edges of G_A are in $G_A[C]$. We need the following lemma.

Lemma 2.5. *Let G be an edge-weighted split graph with a partition (C, I) of $V(G)$, where C and I are a clique and an independent set of G, respectively. If the weighted edges are only in $G[C]$ and the maximum edge weight is $w_{\max}$, then an edge-unweighted split graph G' satisfying $stc(G) = stc(G')$ can be obtained from G in $O(w_{\max} \cdot |E(G)|)$ time.*

Proof. Let $\{u, v\} \in E(G)$ be an edge of weight $w = wei(\{u, v\}) > 1$. We replace the weighted edge between u and v by an unweighted edge, and add $w - 1$ unweighted u–v paths of length two, where the inner point of ith path is a new vertex $x_i \in I$ (see **Fig. 3**). Let us call the obtained graph H. Obviously, this can be done in $O(w)$ time, H has less weighted edges than G, and $stc(H) = stc(G)$ by Lemma 1.2. Also it is easy to see that H is a split graph with weighted edges only in $H[C]$. Therefore, repeatedly applying this local modification, we eventually obtain the desired graph G' in $O(w_{\max} \cdot |E(G)|)$ time. □

Observe that the maximum edge-weight in G_A is bounded by a polynomial function on B and m. Thus the above lemma implies that from an instance A of 3-Partition, we can construct in polynomial time an unweighted split graph G'_A and $k \in \mathbb{Z}^+$ such that A is a yes instance if and only if $stc(G'_A) \le k$. This proves Theorem 2.1.

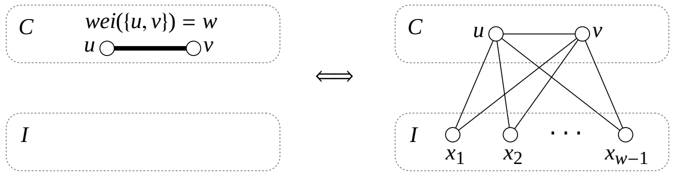

Fig. 3 Weighted edge in the clique C.

2.2 Hardness for chain graphs

Next we prove the NP-hardness for chain graphs. Given an instance A of 3-Partition, we construct the graph $G_A = (P, Q; E)$. For convenience, let $M = B + 3m - 4$ and $\gamma_i = |\{a_j \in A \mid a_j \ge i\}|$. Note that $0 < \gamma_i \le 3m$ for $m + 1 \le i \le a_{3m}$. In particular, $\gamma_{m+1} = 3m$ and $\gamma_{a_{3m}} > 0$. First we define the vertex sets $P = U \cup V \cup W$ and $Q = X \cup Y \cup Z$ as follows:

$$U = \{u_i \mid 1 \le i \le m\}, \qquad X = \{x_i \mid 1 \le i \le 3m\},$$
$$V = \{v_i \mid m + 1 \le i \le a_{3m}\}, \qquad Y = \{y_i \mid m + 1 \le i \le a_{3m}\},$$
$$W = \{w_i \mid 1 \le i \le M - a_{3m}\}, \qquad Z = \{z_i \mid 1 \le i \le M - a_{3m}\}.$$

Next we define the edge set as follows:[⋆1]

$$\begin{aligned} E = {} & (X \times U) \cup (Y \times (U \cup V)) \cup (Z \times (U \cup V \cup W)) \\ & \cup \{\{x_i, v_j\} \mid x_i \in X, m + 1 \le j \le a_i\} \\ & \cup \{\{y_i, w_j\} \mid y_i \in Y, 1 \le j \le M - a_{3m} - \gamma_i\}. \end{aligned}$$

See **Fig. 4** for a simplified illustration of G_A.

Let G_0 and G_1 be two disjoint copies of G_A. That is, $G_A \simeq G_0 \simeq G_1$ and $V(G_0) \cap V(G_1) = \emptyset$. By P_i, Q_i, U_i, V_i, W_i, X_i, Y_i, and Z_i, we denote the vertex sets of G_i, $i \in \{0, 1\}$, that correspond to the vertex sets P, Q, U, V, W, X, Y, and Z of G_A, respectively. Similarly, we denote the vertices of G_i, $i \in \{0, 1\}$, that correspond to vertices u_j, v_j, w_j, x_j, y_j, and z_j of G_A by u^i_j, v^i_j, w^i_j, x^i_j, y^i_j, and z^i_j, respectively. We define the graph H_A as follows (see Fig. 4): $V(H_A) = V(G_0) \cup V(G_1)$ and $E(H_A) = E(G_0) \cup E(G_1) \cup (P_0 \times P_1)$.

Lemma 2.6. *The graph H_A is a chain graph.*

⋆1 For simplicity, we denote by $S \times T$ the set of unordered pairs $\{\{s, t\} \mid s \in S, t \in T\}$.

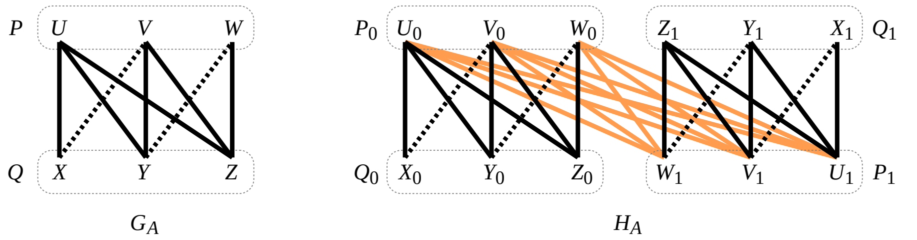

Fig. 4 Graphs G_A and H_A. A solid line between two sets implies that the two sets induce a complete bipartite graph, and a dotted line between two sets implies that there are some (but not all) edges between the two sets. Two color classes of H_A are $P_0 \cup Q_1$ and $Q_0 \cup P_1$.

Proof. Observe that in H_A the following relations hold:

$$N_{H_A}(x_1^0) \subseteq \cdots \subseteq N_{H_A}(x_{3m}^0) \subseteq N_{H_A}(y_1^0) \subseteq \cdots \subseteq N_{H_A}(y_{a_{3m}}^0)$$
$$\subseteq N_{H_A}(z_1^0) = \cdots = N_{H_A}(z_{M-a_{3m}}^0) \subseteq N_{H_A}(w_{M-a_{3m}}^1) \subseteq \cdots \subseteq N_{H_A}(w_1^1)$$
$$\subseteq N_{H_A}(v_{a_{3m}}^1) \subseteq \cdots \subseteq N_{H_A}(v_1^1) \subseteq N_{H_A}(u_m^1) = \cdots = N_{H_A}(u_1^1).$$

This ordering shows that H_A is a chain graph. □

Lemma 2.7. *$deg_{H_A}(u_j^i) = 2M + 2m$, $deg_{H_A}(v_j^i) = 2M - m + \gamma_j > 2M - m$, $2M - a_{3m} \leq deg_{H_A}(w_j^i) \leq 2M - m$, $deg_{H_A}(x_j^i) = a_i$, $deg_{H_A}(y_j^i) = M - \gamma_j < M$, and $deg_{H_A}(z_j^i) = M$. Moreover, $\Delta(H_A) = 2M + 2m$ and $\delta(H_A) = a_1$.* □

Now we prove that A is a yes instance of 3-Partition if and only if $stc(H_A) \leq k$. We divide the proof into the only-if-part (Lemma 2.8) and the if-part (Lemma 2.9).

Lemma 2.8. *Let $k = 3M - m - 2$. If A is a yes instance, then $stc(H_A) \leq k$.*

Proof. Let $A_1, \ldots, A_m$ be the partition of A such that $|A_i| = 3$ and $\sum_{a_j \in A_i} a_j = B$ for $1 \leq i \leq m$. We shall show that there is a spanning tree T of H_A such that $cng_{H_A}(T) \leq k$. Roughly speaking, T is constructed as follows (see **Fig. 5**).

- Take all edges incident to u_1^0 in H_A.
- Take $\{u_j^1, x_h^1\}$ if and only if $a_h \in A_j$.
- Take a perfect matching between $\{u_2^0, \ldots, u_m^0\}$ and $\{x_2^0, \ldots, x_m^0\}$.
- Take a perfect matching between V_i and Y_i for each $i \in \{0, 1\}$.
- Take a perfect matching between W_i and Z_i for each $i \in \{0, 1\}$.

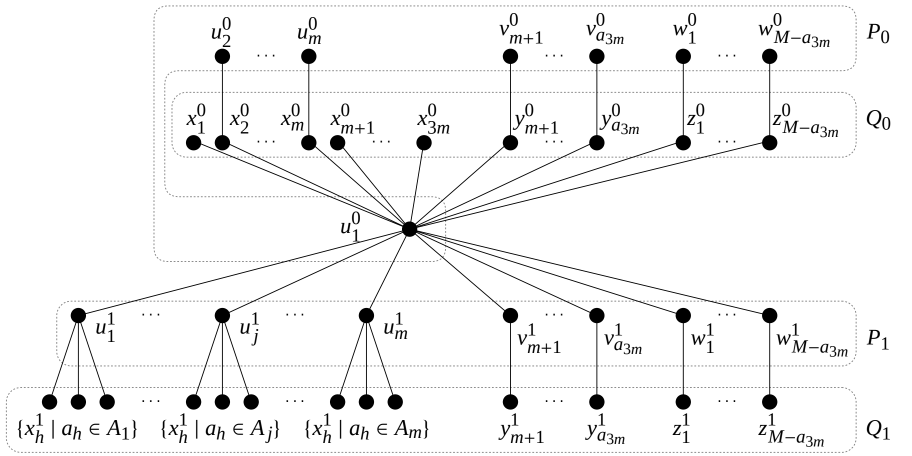

Fig. 5 Spanning tree T in the proof of Lemma 2.8.

More precisely, the edge set of T is defined as follows:

$$E(T) = \{\{u_1^0, v\} \mid v \in Q_0 \cup P_1\} \cup \{\{u_j^1, x_h^1\} \mid a_h \in A_j,\ 1 \leq j \leq m\}$$
$$\cup \{\{u_j^0, x_j^0\} \mid 2 \leq j \leq m\} \cup \{\{v_j^i, y_j^i\} \mid i \in \{0,1\},\ m+1 \leq j \leq a_{3m}\}$$
$$\cup \{\{w_j^i, z_j^i\} \mid i \in \{0,1\},\ 1 \leq j \leq M - a_{3m}\}.$$

Clearly, T is a spanning tree of H_A. It is easy to see that edges not incident to u_1^0 are leaf edges. The congestions of these edges are at most $\Delta(H_A) = 2M + 2m$. Since $B \geq 8$, it follows that $2M + 2m = k - B + 6 < k$. There are four types of inner edges, and they divide $V(H_A)$ as follows.

(1) $\{u_1^0, u_j^1\}$ divides $V(H_A)$ into $\{u_j^1\} \cup \{x_h^1 \mid a_h \in A_j\}$ and its complement.
(2) $\{u_1^0, x_j^0\}$, $2 \leq j \leq m$, divides $V(H_A)$ into $\{u_j^0, x_j^0\}$ and its complement.
(3) $\{u_1^0, v_j^1\}$ or $\{u_1^0, y_j^0\}$ divides $V(H_A)$ into $\{v_j^i, y_j^i\}$ and its complement.
(4) $\{u_1^0, w_j^1\}$ or $\{u_1^0, z_j^0\}$ divides $V(H_A)$ into $\{w_j^i, z_j^i\}$ and its complement.

Hence, it suffices to show that all $|\theta_{H_A}(\{u_j^1\} \cup \{x_h^1 \mid a_h \in A_j\})|$, $|\theta_{H_A}(\{u_j^0, x_j^0\})|$, $|\theta_{H_A}(\{v_j^i, y_j^i\})|$, and $|\theta_{H_A}(\{w_j^i, z_j^i\})|$ are at most k. Note that $\{u_j^1\} \cup \{x_h^1 \mid a_h \in A_j\}$ induces a star $K_{1,3}$, and all $\{u_j^0, x_j^0\}$, $\{v_j^i, y_j^i\}$, and $\{w_j^i, z_j^i\}$ induce edges in H_A. Recall that $M = B + 3m - 4$ and $k = 3M - m - 2$. Thus $2M + 2m = k - B + 6$. *(1)* Since $deg_{H_A}(u_j^1) = 2M + 2m = k - B + 6$ and $\sum_{a_h \in A_j} deg_{H_A}(x_h^1) = B$, we have $|\theta_{H_A}(\{u_j^1\} \cup \{x_h^1 \mid a_h \in A_j\})| = (k - B + 6) + B - 6 = k$.

(2) Since $deg_{H_A}(u_j^0) = 2M+2m = k-B+6$ and $deg_{H_A}(x_j^0) = a_j$, we have $|\theta_{H_A}(\{u_j^0, x_j^0\})| = (k-B+6)+a_j-2 = k-B+a_j+4$. Assumptions $a_j < B/2$ and $B \geq 8$ imply that $|\theta_{H_A}(\{u_j^0, x_j^0\})| \leq k$. *(3)* Since $deg_{H_A}(v_j^i) = 2M-m+\gamma_j$ and $deg_{H_A}(y_j^i) = M-\gamma_j$, we have $|\theta_{H_A}(\{v_j^i, y_j^i\})| = 3M-m-2 = k$. *(4)* Since $deg_{H_A}(w_j^i) \leq 2M-m$ and $deg_{H_A}(z_j^i) = M$, we have $|\theta_{H_A}(\{w_j^i, z_j^i\})| \leq 3M-m-2 = k$. □

Lemma 2.9. *Let $k = 3M-m-2$. If $stc(H_A) \leq k$, then A is a yes instance.*

Proof. Omitted. □

3. Exponential-time exact algorithm

We have shown that STC is NP-complete even for very simple graphs. It is widely believed that NP-hard problems cannot be solved in polynomial time. Thus we need *fast* exponential-time (or sub-exponential-time) algorithms for these problems. Nowadays, designing fast exponential-time exact algorithms becomes an important topic in theoretical computer science. See a recent textbook of exponential-time exact algorithms by Fomin and Kratsch[8]. For STC, we can easily design an $O^*(2^m)$- or $O^*(n^n)$-time algorithm that examine all spanning trees of input graphs, where n and m denote the number of vertices and the number of edges, respectively. In this section, we describe an algorithm for STC that runs in $O^*(2^n)$ time. Although it is still an exponential-time algorithm, it is significantly faster than a naive algorithm.

Let $G = (V, E)$ be a given undirected graph. For convenience, we denote $|\theta_G(X)|$ by $c(X)$. Note that $c(\emptyset) = c(V) = 0$. Consider a spanning tree T with congestion at most k. We regard T as a rooted tree with root $r \in V$. We denote this rooted tree by (T, r). Let $e = \{u, v\} \in E(T)$ be an edge of T, and without loss of generality, let u be the parent of v. Then, the congestion of e in T is equal to $c(D_{T,r}(v))$, where $D_{T,r}(v)$ denotes the set of descendants of v in (T, r). Since the congestion of T is at most k, we see that $c(D_{T,r}(v)) \leq k$. See **Fig. 6**. Conversely, if $c(D_{T,r}(v)) \leq k$ for all $v \in V \setminus \{r\}$, then the congestion of T is at most k. This is because there exists a one-to-one correspondence between the edges e of T and the vertices v in $V \setminus \{r\}$ so that v is a deeper endpoint of e. We summarize this observation in the following lemma.

Lemma 3.1. *The congestion of a rooted tree (T, r) is at most k if and only if $c(D_{T,r}(v)) \leq k$ for every vertex $v \in V \setminus \{r\}$.* □

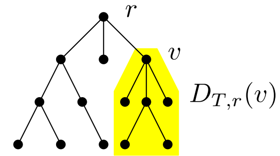

Fig. 6 The definition of $D_{T,r}(v)$.

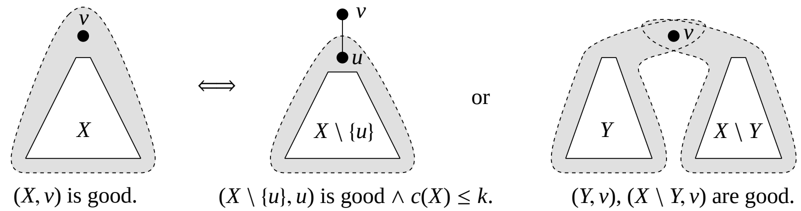

Fig. 7 An illustration of Lemma 3.2.

The lemma above suggests the following dynamic-programming approach. We call a pair (X, v) of a subset $X \subseteq V$ and a vertex $v \notin X$ a *rooted subset* of V. By definition, $X \neq V$ for a rooted subset (X, v) of V. A rooted subset (X, v) of V is *good* if there exists a rooted spanning tree (T, v) of $G[X \cup \{v\}]$ such that $c(D_{T,v}(u)) \leq k$ for all $u \in X$. Here, c is a cut function of G, not of $G[X \cup \{v\}]$. By definition (X, v) is good when $X = \emptyset$. Note that there exists a rooted spanning tree (T, r) of G with congestion at most k if and only if the rooted set $(V \setminus \{r\}, r)$ is good.

The following lemma provides a recursive formula that forms a basis of our algorithm (see **Fig. 7**).

Lemma 3.2. *Let (X, v) be a rooted subset of V with $|X| \geq 1$. Then, (X, v) is good if and only if at least one of the following holds.*

(1) *There exists a vertex $u \in X \cap N_G(v)$ such that $c(X) \leq k$ and $(X \setminus \{u\}, u)$ is good.*

(2) *There exists a non-empty proper subset $Y \subseteq X$ such that both of $(Y, v), (X \setminus Y, v)$ are good.*

Proof. Omitted. □

Lemmas 3.1 and 3.2 above readily give an $O^*(3^n)$-time dynamic programming algorithm. However, the fast subset convolution method enables us to solve the problem in

$O^*(2^n)$ time. We give a more detail below.

Let S be a finite set. For two functions $f, g\colon 2^S \to \mathbb{R}$, their *subset convolution* is a function $f * g\colon 2^S \to \mathbb{R}$ defined as

$$(f * g)(X) = \sum_{Y \subseteq X} f(Y)g(X \setminus Y)$$

for every $X \subseteq S$. Given $f(X), g(X)$ for all $X \subseteq S$, we can compute $(f * g)(X)$ for all $X \subseteq S$ in $O^*(2^n)$ total time, where $n = |S|$[1].

Back to the spanning tree congestion problem, let $v \in V$ be an arbitrary vertex. We define the function $f_v\colon 2^{V\setminus\{v\}} \to \mathbb{R}$ by the following recursion: $f_v(X) = 1$ if $X = \emptyset$; otherwise,

$$f_v(X) = \sum_{u \in X \cap N_G(v)} f_u(X \setminus \{u\}) \max\{0, k - c(X) + 1\} + \sum_{\emptyset \neq Y \subsetneq X} f_v(Y) f_v(X \setminus Y),$$

where the empty sum is defined to be 0. It is easy to verify that $f_v(X)$ is non-negative for every $v \in V$ and every $X \subseteq V \setminus \{v\}$.

The following lemma connects the functions f_v, $v \in V$ and good rooted sets.

Lemma 3.3. *Let (X, v) be a pair of a subset $X \subseteq V \setminus \{v\}$ and a vertex $v \in V$. Then, $f_v(X) > 0$ if and only if (X, v) is a good rooted subset of V.*

Proof. Omitted. □

To apply the subset convolution method, we use the following functions. For each $i \in \{0, 1, \ldots, n-1\}$, where $n = |V|$, and $v \in V$, let $f_v^i\colon 2^{V\setminus\{v\}} \to \mathbb{R}$ be defined by

$$f_v^i(X) = \begin{cases} f_v(X) & \text{if } |X| \leq i, \\ 0 & \text{if } |X| > i, \end{cases}$$

for all $X \subseteq V \setminus \{v\}$. Then, it is not difficult to see the following.

(1) For all $v \in X$ and $X \subseteq V \setminus \{v\}$, $f_v^{n-1}(X) = f_v(X)$.

$$f_v^{n-1}(X) = f_v(X).$$

(2) For all $v \in V$ and $X \subseteq V \setminus \{v\}$,

$$f_v^0(X) = \begin{cases} 1 & \text{if } X = \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

(3) For all $i \in \{1, \ldots, n-1\}$, $v \in V$, and $X \subseteq V \setminus \{v\}$

$$\begin{aligned} f_v^i(X) &= \sum_{u \in X \cap N_G(v)} f_u^{i-1}(X \setminus \{u\}) \max\{0, k - c(X) + 1\} \\ &\quad + \sum_{\emptyset \neq Y \subsetneq X} f_v^{i-1}(Y) f_v^{i-1}(X \setminus Y) \\ &= \sum_{u \in X \cap N_G(v)} f_u^{i-1}(X \setminus \{u\}) \max\{0, k - c(X) + 1\} \\ &\quad + \sum_{Y \subseteq X} f_v^{i-1}(Y) f_v^{i-1}(X \setminus Y) - 2 f_v^{i-1}(\emptyset) f_v^{i-1}(X) \\ &= \sum_{u \in X \cap N_G(v)} f_u^{i-1}(X \setminus \{u\}) \max\{0, k - c(X) + 1\} \\ &\quad + (f_v^{i-1} * f_v^{i-1})(X) - 2 f_v^{i-1}(\emptyset) f_v^{i-1}(X). \end{aligned}$$

Our algorithm is based on these formulas.

Step 1. For all $v \in V$ and $X \subseteq V \setminus \{v\}$, compute $f_v^0(X)$ based on the formulas above.

Step 2. For each $i = 1, \ldots, n-1$ in the ascending order, do the following.

Step 2-1. For all $v \in V$, compute the subset convolution $f_v^{i-1} * f_v^{i-1}$.

Step 2-2. For all $v \in V$ and all $X \subseteq V \setminus \{v\}$, compute $f_v^i(X)$ based on the formula above.

Step 3. If $f_v^{n-1}(V) > 0$, then output Yes. Otherwise, output No.

The correctness is immediate from the discussion so far. The running time is $O^*(2^n)$ since the running time of each step is bounded by $O^*(2^n)$. This is an algorithm for solving the decision problem, but a simple binary search on $k \in \{1, \ldots, |E|\}$ can provide the spanning tree congestion. Thus, we obtain the following theorem.

Theorem 3.4. *The spanning tree congestion of a given undirected graph can be computed in $O^*(2^n)$ time.*

Note that the algorithm also works for the weighted case with the $O(n)$-factor increase of the running time, since the number of distinct cut values $c(X)$ is bounded by 2^n and so the binary search over the all possible values of $c(X)$ takes at most $O(\log(2^n)) = O(n)$ iterations. This is possible if we compute $c(X)$ for all $X \subseteq V$ beforehand, which only takes $O^*(2^n)$ time.

4. Remarks on cographs

We showed NP-completeness of STC for graphs of clique-width at most three. Therefore, it is quite natural to ask whether or not STC is NP-complete for graphs of clique-width at most two; that is, for cographs[7]. Although the complexity of STC for cographs remains unsettled, we have the following theorem.

Theorem 4.1. *The spanning tree congestion of cographs can be approximated within a factor three in polynomial time. Furthermore, the spanning tree congestion of chordal cographs can be determined in linear time.*

Let $\mu_G(u, v)$ be the maximum number of edge-disjoint u–v paths in G, and let $\mu(G) = \max_{u,v \in V(G)} \mu_G(u, v)$. Ostrovskii[19] showed that $stc(G) \geq \mu(G)$ for any graph G. Let G be a connected cograph with at least two vertices. Then G can be expressed as $G_1 \oplus G_2$ for some nonempty cographs G_1 and G_2. Recall that a vertex $v \in V(G)$ is universal if v is adjacent to all other vertices in G.

Lemma 4.2. *The spanning tree congestion of a graph with a universal vertex can be determined in linear time.*

Proof. Omitted. □

Let $G = G_1 \oplus G_2$ is a chordal cograph. We can assume that one of G_1 and G_2 is K_1[23]. Therefore, every connected chordal cograph with at least two vertices has a universal vertex. Thus we have the second part of Theorem 4.1. The first part of Theorem 4.1 is a corollary to the following lemma.

Lemma 4.3. *The spanning tree congestion of $G = G_1 \oplus G_2$ can be approximated within a factor three in polynomial time.*

Proof. Omitted. □

References

1) A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: Fast subset convolution. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC '07)*, pages 67–74, 2007.
2) H.L. Bodlaender, F.V. Fomin, P.A. Golovach, Y. Otachi, and E.J. van Leeuwen. Parameterized complexity of the spanning tree congestion problem. submitted.
3) H.L. Bodlaender, K. Kozawa, T. Matsushima, and Y. Otachi. Spanning tree congestion of k-outerplanar graphs. *Discrete Math.*, 311:1040–1045, 2011.
4) A.Brandstädt, V.B. Le, and J.P. Spinrad. *Graph Classes: A Survey*. SIAM, 1999.
5) A.Brandstädt and V.V. Lozin. On the linear structure and clique-width of bipartite permutation graphs. *Ars Combin.*, 67:273–281, 2003.
6) A.Castejón and M.I. Ostrovskii. Minimum congestion spanning trees of grids and discrete toruses. *Discuss. Math. Graph Theory*, 29:511–519, 2009.
7) B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Appl. Math.*, 101:77–114, 2000.
8) F.V. Fomin and D.Kratsch. *Exact Exponential Algorithms*. Springer, 2010.
9) M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.
10) M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*, volume 57 of *Annals of Discrete Mathematics*. North Holland, second edition, 2004.
11) P.Hliněný, S.Oum, D.Seese, and G.Gottlob. Width parameters beyond tree-width and their applications. *Comput. J.*, 51:326–362, 2008.
12) S.W. Hruska. On tree congestion of graphs. *Discrete Math.*, 308:1801–1809, 2008.
13) K.Kozawa and Y.Otachi. Spanning tree congestion of rook's graphs. *Discuss. Math. Graph Theory*, to appear.
14) K. Kozawa, Y. Otachi, and K. Yamazaki. On spanning tree congestion of graphs. *Discrete Math.*, 309:4215–4224, 2009.
15) H.-F. Law. Spanning tree congestion of the hypercube. *Discrete Math.*, 309:6644–6648, 2009.
16) H.-F. Law and M.I. Ostrovskii. Spanning tree congestion: duality and isoperimetry; with an application to multipartite graphs. *Graph Theory Notes N. Y.*, 58:18–26, 2010.
17) H.-F. Law and M.I. Ostrovskii. Spanning tree congestion of some product graphs. *Indian J. Math.*, to appear.
18) C.Löwenstein, D.Rautenbach, and F.Regen. On spanning tree congestion. *Discrete Math.*, 309:4653–4655, 2009.
19) M.I. Ostrovskii. Minimal congestion trees. *Discrete Math.*, 285:219–226, 2004.
20) M.I. Ostrovskii. Minimum congestion spanning trees in planar graphs. *Discrete Math.*, 310:1204–1209, 2010.
21) Y. Otachi, H.L. Bodlaender, and E.J. van Leeuwen. Complexity results for the spanning tree congestion problem. In *WG 2010*, volume 6410 of *Lecture Notes in Comput. Sci.*, pages 3–14. Springer-Verlag, 2010.
22) S.Simonson. A variation on the min cut linear arrangement problem. *Math. Syst. Theory*, 20:235–252, 1987.
23) J.-H. Yan, J.-J. Chen, and G.J. Chang. Quasi-threshold graphs. *Discrete Appl. Math.*, 69:247–255, 1996.
24) M.Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.*, 2:77–79, 1981.