Simpler Computation of Minimal Removable Sets in 2-Edge-Connected Systems for Undirected Graphs

SHOTA, Kan HARAGUCHI, Kazuya

30th December 2025

1 Introduction

2 Preliminaries

2.1 Graphs

Throughout the paper, we assume that a graph is simple and undirected. We also assume that each vertex v is associated with a fixed integer identifier $id(v)$. These identifiers are given as part of the input graph and are preserved in all induced subgraphs.

For a graph $G = (V, E)$ with a vertex set V and an edge set E, we may abbreviate an edge $\{u, v\} \in E$ as uv (or equivalently vu) for simplicity. For a vertex $v \in V$, we denote by $N_G(v)$ the set of neighbours of v. Let $S \subseteq V$ be a subset of vertices. We denote by $N_G(S) = \bigcup_{v \in S} N_G(v) \setminus S$, the set of open neighbours of S in G. We denote by $G[S]$ the subgraph of G induced by S. We denote by $G - S$ the subgraph obtained by removing S and all edges to incident to S, that is, $G - S = G[V - S]$. If $S = \{v\}$, then we simply write $G - v$ instead of $G - \{v\}$. A *degree-two segment* in G is a maximal path whose vertices all have degree two in G. Note that a degree-two segment may consist of a single vertex.

2.2 DFS-Trees and Related Notions

A *DFS-tree* of a connected undirected graph $G = (V, E)$ is a spanning tree T obtained by performing a depth-first search (DFS) on G. We root T at an arbitrary vertex $r \in V$. For each $v \in V$, we denote by T_v the subtree of T rooted at v.

An edge in E is called a *tree edge* if it appears in T, and a *back edge* otherwise. For $u, v \in V$, we say that u is an *ancestor* of v if u lies on the unique r–v path in T. Moreover, if u is an ancestor of v and $u \neq v$, then we say that u is a *proper ancestor* of v, and v is correspondingly a (*proper*) *descendant* of u. If uv is a tree edge and u is an ancestor of v, then u is the *parent* of v; we denote this by $\pi(v) = u$.

We may orient edges based on the DFS-tree T: for a tree edge $\pi(v)v$, we orient it from $\pi(v)$ to v; and for a back edge xy, we orient it from the descendant to the ancestor. We may write (u, v) and (x, y) to indicate the orientation of edge uv and xy, and refer to it as the *outgoing edge* from u or x and the *incoming edge* to v or y.

We say that a tree edge $\pi(v)v$ is *covered* by a back edge $e = (x, y)$ if x is a descendant of v and y is a proper ancestor of v (an ancestor of $\pi(v)$). In this case, e together with $\pi(v)v$ forms part of a cycle in G. Note that every tree edge is covered by at least one back edge if G is 2-edge-connected. For a tree edge uv, we denote by $Cov(uv) = \{(x, y) \in E \mid (x, y) \text{ is a back edge covering } uv\}$ the set of back edges that cover uv. We refer to $CovMult(uv) \triangleq |Cov(uv)|$ as the *cover multiplicity* of the tree edge uv.

In addition to cover multiplicity, we also consider aggregated identifiers of endpoints of covering back edges. For a tree edge uv and $i \in \{1, 2\}$, we define

$$AggDesc_i(uv) \triangleq \sum_{(x,y)\in Cov(uv)} (id(x))^i\,, \qquad AggAnc_i(uv) \triangleq \sum_{(x,y)\in Cov(uv)} (id(y))^i\,.$$

We refer to these quantities as *aggregated identifiers*.

2.3 Graph Contraction and Suppression

Let $G = (V, E)$ be a graph. For $S \subseteq V$, the *contraction* of S in G is the graph obtained by merging all vertices in S into a single vertex s, removing all edges with both endpoints in S, and replacing each edge with one endpoint in S by an edge incident to s. For a vertex $v \in V$ with degree two, the *suppression* of v in G is the graph obtained by removing v and its two incident edges, and adding a new edge between the two neighbours of v. Equivalently, assuming that v has two neighbours u and w, we contract the set $\{u, v\}$ or $\{v, w\}$ in G.

3 Cover Information on DFS-Trees

For later use, we present efficient algorithms for computing various cover-related quantities in this section. In Section 3.1, we introduce a general technique, called *difference-based aggregation*, which efficiently supports path-based update operations on a rooted tree. In Section 3.2, we apply this technique to compute various cover-related quantities for all tree edges in a 2-edge-connected undirected graph in $O(n + m)$ time.

Before presenting the algorithms, we briefly illustrate how aggregated identifiers can be used to detect structural properties of covering back edges. For a tree edge uv, we can determine whether all covering back edges outgo from the same descendant or ingo to the same ancestor using the aggregated identifiers. In order to show this, we first prove a general lemma on sequences of real numbers.

Lemma 1 *Let $(\alpha_1, \alpha_2, \ldots, \alpha_q)$ be a sequence of q real numbers. Then, all elements in the sequence are equal, i.e., $\alpha_1 = \alpha_2 = \cdots = \alpha_q$, if and only if*

$$\left(\sum_{i=1}^{q} \alpha_i\right)^2 = q \sum_{i=1}^{q} (\alpha_i)^2\,.$$

Proof: We have

$$
\begin{aligned}
q\sum_{i=1}^{q}(\alpha_i)^2 - \left(\sum_{i=1}^{q}\alpha_i\right)^2 &= \sum_{i=1}^{q}(q-1)\,(\alpha_i)^2 - \sum_{i=1}^{q-1}\sum_{j=i+1}^{q} 2\alpha_i\alpha_j \\
&= \sum_{i=1}^{q}(q-i+i-1)\,(\alpha_i)^2 - \sum_{i=1}^{q-1}\sum_{j=i+1}^{q} 2\alpha_i\alpha_j \\
&= \sum_{i=1}^{q-1}\sum_{j=i+1}^{q}(\alpha_i)^2 + \sum_{i=2}^{q}\sum_{j=1}^{i-1}(\alpha_i)^2 - \sum_{i=1}^{q-1}\sum_{j=i+1}^{q} 2\alpha_i\alpha_j \\
&= \sum_{i=1}^{q-1}\sum_{j=i+1}^{q}\left[(\alpha_i)^2 - 2\alpha_i\alpha_j + (\alpha_j)^2\right] = \sum_{i=1}^{q-1}\sum_{j=i+1}^{q}(\alpha_i-\alpha_j)^2 .
\end{aligned}
$$

Since each term in the last summation is nonnegative, the entire summation is equal to zero if and only if $\alpha_i = \alpha_j$ for all $i, j \in \{1, \ldots, q\}$. □

Using Lemma 1, we can prove the following lemma on aggregated identifiers.

Lemma 2 *Let $G = (V, E)$ be a 2-edge-connected undirected graph, T be a DFS-tree of G rooted at $r \in V$, and uv be a tree edge in T. Moreover, let $Cov(uv) = \{(x_1, y_1), \ldots, (x_q, y_q)\}$ be the set of back edges covering uv, where $q = CovMult(uv)$. Then, the following two conditions hold:*

- $x_1 = x_2 = \cdots = x_q$ *if and only if* $(AggDesc_1(uv))^2 = q \cdot AggDesc_2(uv)$*; and*
- $y_1 = y_2 = \cdots = y_q$ *if and only if* $(AggAnc_1(uv))^2 = q \cdot AggAnc_2(uv)$*.*

Proof: The conditions follow directly from Lemma 1 by setting $\alpha_i := id(x_i)$ for the descendant-side condition and $\alpha_i := id(y_i)$ for the ancestor-side condition. □

3.1 Difference-Based Aggregation on a Rooted Tree

Let T be a rooted tree. Each edge e stores a value, say $A(e)$ (from $\mathbb{Z}$, $\mathbb{R}$, or, more generally, from an additive abelian group), and update operations add a constant value to all edges on a specified ancestor–descendant path in T.

Formally, update operations are specified by triples (x, y, α), where y is an ancestor of x in T, and α is a value to be added to all edges on the x–y path. The goal is to compute the final value stored in each edge after performing a sequence of such update operations.

In order to achieve this goal efficiently, we use a difference-based approach. We associate each vertex v in T with an auxiliary value $\Delta(v)$, initially set to zero. For each update operation (x, y, α), we perform:

$$\Delta(x) := \Delta(x) + \alpha, \qquad \Delta(y) := \Delta(y) - \alpha.$$

Algorithm 1 Difference-Based Aggregation on a Rooted Tree

Input: A rooted tree T, an initial value $A(e)$ for all edges e in $E(T)$, and a sequence of update operations (x, y, α)

Output: The final values for all edges

Initialise $\Delta(v) := 0$ for all $v \in V(T)$ and $\mathit{value}(e) := A(e)$ for all $e \in E(T)$;
for each update operation (x, y, α) **do**
    $\Delta(x) := \Delta(x) + \alpha$; $\Delta(y) := \Delta(y) - \alpha$
end for;
for each child w of the root r in T **do**
    Execute AGGREGATE-DFS(w)
end for;
output $\mathit{value}(e)$ for all edges e in $E(T)$

procedure AGGREGATE-DFS(v)
    $\mathit{value}(\pi(v)v) := \mathit{value}(\pi(v)v) + \Delta(v)$;
    for each child w of v in T **do**
        Execute AGGREGATE-DFS(w);
        $\mathit{value}(\pi(v)v) := \mathit{value}(\pi(v)v) + \mathit{value}(vw)$
    end for
end procedure

Lemma 3 *Let T be a rooted tree and each edge e store an initial value $A(e)$. After performing a sequence of update operations represented as triples (x, y, α) on a rooted tree T, for each edge $\pi(v)v$, the updated value is given by*

$$A(\pi(v)v) + \sum_{w \in V(T_v)} \Delta(w).$$

PROOF: Consider a single update operation (x, y, α). By the definition of Δ, the operation contributes $+\alpha$ to $\Delta(x)$ and $-\alpha$ to $\Delta(y)$. We examine how such an operation contributes to the sum $\sum_{w \in V(T_v)} \Delta(w)$.

If $x \in V(T_v)$ and $y \notin V(T_v)$, then the x–y path passes through $\pi(v)v$, and the operation contributes $+\alpha$ to the sum. If both x and y belong to $V(T_v)$, then the x–y path does not pass through $\pi(v)v$, and its $+\alpha$ and $-\alpha$ contributions cancel within T_v, yielding a total contribution of 0. If neither x nor y is in T_v, then the x–y path does not pass through $\pi(v)v$, and again contributes 0 to the sum. Since y is a proper ancestor of x, the case with $x \notin V(T_v)$ and $y \in V(T_v)$ cannot occur.

Therefore, the only update operations that contribute a nonzero amount are those whose x–y path passes through $\pi(v)v$, and each such operation contributes exactly $+\alpha$. $\square$

Lemma 4 *Given a rooted tree T with n vertices and a sequence of q update operations, Algorithm 1 computes the final values for all edges in $O(q+n)$ time.*

PROOF: By Lemma 3, it suffices to show that Algorithm 1 correctly computes $A(\pi(v)v)+\sum_{w\in V(T_v)}\Delta(w)$ for each edge $\pi(v)v$.

We first show the case with v being a leaf. In this case, since there is no child of v in T, the algorithm sets $\mathit{value}(\pi(v)v) = A(\pi(v)v) + \Delta(v)$.

Next, we consider the case other than the leaf case. We assume that, for any child w of v in T, AGGREGATE-DFS(w) correctly computes the sum of the auxiliary values Δ for all vertices in T_w. Then, the algorithm computes

$$A(\pi(v)v) + \Delta(v) + \sum_{w\text{: child of } v}\ \sum_{z\in V(T_w)} \Delta(z) = A(\pi(v)v) + \sum_{z\in V(T_v)} \Delta(z)$$

that is the sum of the auxiliary values Δ for all vertices in T_v.

Since each update operation is processed in constant time, the total time for all q update operations is $O(q)$. We visit each vertex exactly once, and consider all edges exactly once, which means we can correctly compute the final values for all edges. Since we traverse only edges in T, we can execute the aggregation in $O(n)$ time. □

3.2 Computation of Cover Information

Let $G=(V,E)$ be a 2-edge-connected undirected graph and T be a DFS-tree of G rooted at $r\in V$. As described in Section 2.2, every back edge (x,y) is oriented from a descendant x to a proper ancestor y.

Each back edge (x,y) covers all tree edges on the unique y–x path in T. Therefore, we have a natural mapping from back edges to an update opeation (x,y,α) on T, where α is chosen appropriately depending on the quantity to be computed.

Lemma 5 *Given a 2-edge-connected undirected graph $G=(V,E)$ with $n=|V|$ and $m=|E|$, and a DFS-tree T of G rooted at $r\in V$, we can compute the following quantities for all tree edges in T in $O(n+m)$ time:*

- *Cover multiplicity $\mathit{CovMult}(uv)$;*
- *Aggregated identifiers $\mathit{AggDesc}_1(uv)$ and $\mathit{AggDesc}_2(uv)$; and*
- *Aggregated identifiers $\mathit{AggAnc}_1(uv)$ and $\mathit{AggAnc}_2(uv)$.*

PROOF: By the definiton of each quantity, we can represent the contribution of each back edge (x,y) as an update operation (x,y,α) on T, where α is set as follows:

- For $\mathit{CovMult}(uv)$, set $\alpha := 1$;
- For $\mathit{AggDesc}_i(uv)$, set $\alpha := (id(x))^i$ for $i\in\{1,2\}$; and

- For $AggAnc_i(uv)$, set $\alpha := (id(y))^i$ for $i \in \{1, 2\}$.

Since there are $O(m)$ back edges in G, we can perform $O(m)$ update operations on T. By Lemma 4, we can compute the final values for all tree edges in $O(n+m)$ time. □

4 Mathematical Properties and Computation of Minimal Removable Sets

In this section, we study structural properties of MinRSs and present an efficient algorithm for computing all MinRSs in a 2-edge-connected undirected graph.

MinRSs in a 2-edge-connected undirected graph have been studied in [1, 2]. In these works, they showed the following necessary condition for MinRSs.

Lemma 6 (Observation 3 in [1] and Lemma 7 in [2]) *Let $G = (V, E)$ be a 2-edge-connected undirected graph with $|V| \geq 3$. If a nonempty proper subset $S \subsetneq V$ is a MinRS of G, then,*

- *S forms a degree-two segment in G; or*
- *S is a singleton $\{v\}$, where $v \in V$.*

By Lemma 6, we can list all candidates for MinRSs in $O(n+m)$ time by finding all degree-two segments in G and collecting all singleton vertex sets.

Based on Lemma 6, we classify MinRSs into several types to make them easier to handle with a DFS-tree T of G rooted at an arbitrary vertex $r \in V$:

- the MinRS containing the root of the DFS-tree;
- MinRSs containing at least one leaf of the DFS-tree;
- MinRSs consisting only of internal vertices of the DFS-tree that form a degree-two segment in G; and
- MinRSs consisting of a single internal vertex of the DFS-tree with degree at least three in G.

The MinRS containing the root of the DFS-tree contains no leaf of the DFS-tree; this categorisation is mutually exclusive.

Lemma 7 *Let $G = (V, E)$ be a 2-edge-connected undirected graph with $|V| \geq 3$ and T be a DFS-tree of G rooted at $r \in V$. We can determine whether G has a MinRS containing r and find such a MinRS in $O(n+m)$ time.*

Proof: By Lemma 6, if G has a MinRS containing r, then it is either $\{r\}$ (if $\deg_G(r) \geq 3$) or a degree-two segment P containing r (otherwise). All we have to do is to check $G - r$ or $G - P$ for 2-edge-connectivity, which can be done in $O(n+m)$ time. □

In the rest of this section, we focus on MinRSs that do not contain the root of the DFS-tree. Before going into details, we show a basic property of degree-two segments not containing the root of the DFS-tree for the sake of simplifying discussions in the following subsections.

Lemma 8 *Let $G = (V, E)$ be a 2-edge-connected undirected graph with $|V| \geq 3$, T be a DFS-tree of G rooted at $r \in V$, and Let $P = v_1 \to v_2 \to \cdots \to v_k$ be a degree-two segment in G that does not contain r. Then, the following hold:*

(i) *No internal vertex of T that is an endpoint of any back edge appears in P; and*

(ii) *The sequence $v_1, v_2, \ldots, v_k$ appears in this order or its reverse order on a root-to-leaf path in T.*

(iii) *If P contains a leaf of T, then it is either v_1 or v_k, but not both.*

Proof: (i) Since every internal vertex of T has the parent and at least one child, if an internal vertex of T is an endpoint of some back edge, then the degree in G is at least three. Thus, such a vertex cannot appear in P.
(ii) In the DFS traversal that constructs T, the vertices in P are visited one by one without visiting any other vertex in between. (iii) is immediate from (ii). □

4.1 MinRSs Containing DFS-Tree Leaf

In this subsection, we consider MinRSs that consist of only leaves of a DFS-tree.

We first show a basic property of leaves in a DFS-tree of a 2-edge-connected undirected graph.

Lemma 9 *Let $G = (V, E)$ be a 2-edge-connected undirected graph with $|V| \geq 3$ and T be a DFS-tree of G rooted at $r \in V$. For any leaf $v \in V$ of T or any degree-two segment P containing a leaf of T, $G - v$ or $G - P$ is connected.*

Proof: Since T is a spanning tree of G, every path in T also exists in G. For any leaf $v \in V$ of T or any degree-two segment P containing a leaf of T, $T - v$ or $T - P$ is connected. Thus, $G - v$ or $G - P$ is also connected. □

Lemma 10 *Let $G = (V, E)$ be a 2-edge-connected undirected graph with $|V| \geq 3$ and T be a DFS-tree of G rooted at $r \in V$. Let $S \subsetneq V$ be a nonempty proper subset of V that is either a singleton consisting of a leaf of T or a degree-two segment containing a leaf of T; let x denote the leaf of T in S and a denote the other endpoint of S. Note that $a = x$ if S is a singleton. Then, S is a MinRS of G if and only if, for any tree edge (u, v) with $v \notin S$, there exists a back edge outgoing from some vertex in $V \setminus S$ that covers (u, v).*

Proof: By Lemma 8, S contains exactly one leaf of T and S appears as a path from some vertex a to its descendant x in T. (If S is a singleton, then $a = x$.) By the definition of S, x is a leaf of T. Since G is 2-edge-connected, every tree edge is covered by at least one back edge. Thus, it suffices to consider only tree edges on the y–$\pi(a)$ path in T, where y is the most distant proper ancestor of x adjacent to x by a back edge.
($\Longrightarrow$) We show by contraposition. Suppose that there exists a tree edge (u, v) on the y–$\pi(a)$ path in T that is not covered by any back edge outgoing from a vertex other than those in S. Any vertex in $T_v - S$ cannot have a back edge to a proper ancestor of u in $G - S$, which means that uv is a bridge in $G - S$.
($\Longleftarrow$) By Lemma 6, it suffices to show that $G-S$ is 2-edge-connected. By Lemma 9, $G-S$ is connected. We show that there is no bridge in $G-S$. Every back edge $(x', y') \in E \setminus E(T)$ with $x', y' \notin S$ is not a bridge in $G - S$, since $x'y'$ and the y'–x' path in T form a cycle in $G-S$. Next, we consider tree edges in $G-S$. For any tree edge not on the y–$\pi(a)$-path in T, no covering back edge disappears by removing S from G, which means that such a tree edge is not a bridge in $G - S$. For any tree edge (u, v) on the y–$\pi(a)$-path in T, by the assumption, there exists a back edge outgoing from a vertex in $T_v - S$ that covers (u, v), which means that such a tree edge is not a bridge in $G - S$. Thus, there is no bridge in $G - S$, and hence, $G - S$ is 2-edge-connected. $\square$

Let $S \subsetneq V$ be a nonempty proper subset of V that is either a singleton consisting of a leaf of T or a degree-two segment containing a leaf of T; let x denote the leaf of T. Since x is the only vertex in S that has back edges outgoing from it, then by Lemma 10, S is a MinRS of G if and only if, for any tree edge (u, v) with $v \notin S$, there exists a back edge outgoing from other vertex than x that covers (u, v).

Theorem 1 *Given a 2-edge-connected undirected graph $G = (V, E)$ with $n = |V|$ and $m = |E|$, Algorithm 2 computes all MinRSs of G containing a leaf of a DFS-tree T in $O(n + m)$ time.*

Proof: Note that we assume that a graph has no parallel edges throughout this paper. If $|V| < 3$, i.e., G consists of only one vertex, then G has no MinRS. Thus, we assume that $|V| \geq 3$. As the given graph is 2-edge-connected, each tree edge is covered by at least one back edge. Let $S \subsetneq V$ be a nonempty proper subset of V that is either a singleton consisting of a leaf of T or a degree-two segment containing a leaf of T; let x denote the leaf of T in S. By the discussion before the theorem, S is a MinRS of G if and only if, for any tree edge (u, v) with $v \notin S$, there exists a back edge outgoing from other vertex than x that covers (u, v); which is equivalent to that for any tree edge (u, v) with $v \notin S$, $X_{uv} = \{x' \in V \mid \text{There is back edge } (x', y') \text{ covering } (u, v)\} \neq \{x\}$ holds. By Lemma 5, we can compute the aggregated identifiers $AggDesc_1$ and $AggDesc_2$ for all tree edges.

By Lemma 5, we can compute $AggDesc_1$ and $AggDesc_2$ for all tree edges in $O(n + m)$ time. For each tree edge (u, v), we can compute the set X_{uv} from $AggDesc_1(uv)$ and $AggDesc_2(uv)$ in constant time. For each leaf l of T, we can check whether S is a MinRS

Algorithm 2 Computation of All MinRSs Containing a Leaf of a DFS-tree

Input: A 2-edge-connected undirected graph $G = (V, E)$ and its DFS-tree T rooted at $r \in V$

Output: All MinRSs of G containing a leaf of T

Compute $\mathit{AggDesc}_1$ and $\mathit{AggDesc}_2$ for all tree edges;
Initialise $\mathit{covering\_sets}_l :=$ an array from leaves l of T to sets of tree edges;
for each tree edge (u, v) in $E(T)$ **do**
  Compute the set $X_{uv} := \{x' \in V \mid \text{There is back edge } (x', y') \text{ covering } (u, v)\}$ from $\mathit{AggDesc}_1(uv)$ and $\mathit{AggDesc}_2(uv)$;
  Append (u, v) to $\mathit{covering\_sets}_l$ **if** X_{uv} is a singleton $\{l\}$ and l is a leaf of T
end for;
Initialise $\mathcal{Y} :=$ an empty set;
for each leaf l of T **do**
  $S := \begin{cases} \{l\} & \text{if } \deg_G(l) \geq 3, \\ \text{the vertex set of the degree-two segment containing } l & \text{otherwise;} \end{cases}$
  IS-MINRS := FALSE;
  for each tree edge (u, v) in $\mathit{covering\_sets}_l$ **do**
    if $u \notin S$ and $v \notin S$ **then**
      IS-MINRS := TRUE ▷ You can break the loop here
    end if
  end for;
  $\mathcal{Y} := \mathcal{Y} \cup \{S\}$ **if** IS-MINRS
end for;
output $\mathcal{Y}$

of G in time proportional to the size of $\mathit{covering\_sets}_l$. Since the total size of $\mathit{covering\_sets}_l$ for all leaves l of T is at most $O(n)$, we can check all candidates S in $O(n)$ time. Thus, the total time complexity is $O(n + m)$. □

4.2 Degree-Two Segment MinRSs but all Internal Vertices

In this section, we consider MinRSs that consist of only internal vertices of a DFS-tree and form a degree-two segment in G.

To simplify discussions, we introduce the *suppression* for degree-two segments. First, we show that the two neighbours of the endpoints of a degree-two segment not containing the root or any leaf of a DFS-tree are distinct.

Lemma 11 *Let $G = (V, E)$ be a 2-edge-connected undirected graph with $|V| \geq 3$, and P be a degree-two segment in G that does not contain the root or any leaf of a DFS-tree T of G. Then, $|N_G(V(P))| = 2$ holds.*

PROOF: Let $P = v_1 \to v_2 \to \cdots \to v_k$, and let a, b denote the neighbours of v_1 and v_k not on P if P is not a singleton; if P is a singleton, then let a and b denote the two neighbours of the only vertex in P. We show that $a \neq b$ holds. By Lemma 8(ii), P appears as the v_1–v_k path in T, and without loss of generality, we assume that v_1 is an ancestor of v_k in T. The DFS traversal visits in order $a, v_1, v_2, \dots, v_k$. If $a = b$, then $v_k b$ is a back edge, which means that b would be a leaf of T since v_k has degree 2 and b is its only neighbour other than v_{k-1}. This contradicts the assumption that P does not contain any leaf of T. $\square$

We fix a degree-two segment $P = v_1 \to v_2 \to \cdots \to v_k$ in G, and let $\{a, b\} = N_G(V(P))$, where $a \neq b$ holds by Lemma 11. We define the *suppression* of P to be the graph G_P obtained from G by removing all vertices of P and adding a new edge $e_P = ab$, which means that the new edge e_P is not a self-loop. Note that G_P or the suppressed DFS-tree T_P may have parallel edges even if G has no parallel edges.

Lemma 12 *Let $G = (V, E)$ be a 2-edge-connected undirected graph with $|V| \geq 3$, T be a DFS-tree of G rooted at $r \in V$, and $P = v_1 \to v_2 \to \cdots \to v_k$ be a degree-two segment in G that does not contain r or any leaf of T; let a and b denote the neighbours of v_1 and v_k not on P. Then, T_P is a DFS-tree of G_P rooted at r, where T_P is the tree obtained from T by surpressing P. Moreover, e_P is a tree edge in T_P and CovMult(e_P) in G_P equals any CovMult(uv) in G for a tree edge (u, v) on the a–b path in T.*

Proof: Let $P = v_1 \to v_2 \to \cdots \to v_k$, and let a, b denote the neighbours of v_1 and v_k not on P. By Lemmas 8(ii) and 11, P appears as the v_1–v_k path in T, and without loss of generality, we assume that v_1 is an ancestor of v_k in T. In the DFS traversal that constructs T, the vertices in order $a, v_1, v_2, \dots, v_k, b$ are visited one by one without visiting any other vertex in between. For DFS traversal on G_P, when we visit a, we can directly visit b via the new edge $e_P = ab$ without visiting any other vertex in between. Thus, T_P is a DFS-tree of G_P rooted at r.
By the construction of T_P, e_P is a tree edge in T_P. Next, we show that *CovMult*(e_P) in G_P equals any *CovMult*(uv) in G for a tree edge (u, v) on the a–b path in T. No vertex in P is an endpoint of any back edge by Lemma 8(i). Thus, any back edge in G corresponds to a back edge in G_P and vice versa. Moreover, a back edge covers e_P in G_P if and only if it covers any tree edge (u, v) on the a–b path in T in G. Thus, the cover multiplicities are equal. $\square$

Lemma 13 *Let $G = (V, E)$ be a 2-edge-connected undirected (multi)graph with $|V| \geq 3$, T be a DFS-tree of G rooted at $r \in V$, and $e = (u, v)$ be a tree edge in T. Then, e belongs to some 2-edge-cut if and only if CovMult$(e) = 1$ in G. Moreover, if e belongs to some 2-edge-cut, then there exists a unique tree edge $f \neq e$ such that $E(V(T_v)) = \{e, f\}$ holds.*

Proof: Let $X := V(T_v)$. ($\Longrightarrow$) Let $\{e, f\}$ be a 2-edge-cut containing e for some tree edge $f \neq e$. In $G - e$, X and $V \setminus X$ are connected separately. Since $G - \{e, f\}$ is disconnected,

the connected components of $G - \{e, f\}$ are exactly X and $V \setminus X$. Thus, f is the only back edge covering e in G, which means that $\mathit{CovMult}(e) = 1$.
($\Longleftarrow$) Suppose that $\mathit{CovMult}(e) = 1$. Let f be the unique back edge covering e in G. In $G - e$, X and $V \setminus X$ are connected separately. Since f is the only back edge covering e, $G - \{e, f\}$ is disconnected. Thus, $\{e, f\}$ is a 2-edge-cut containing e.
Moreover, if e belongs to some 2-edge-cut, then the above argument shows that there exists a unique tree edge $f \neq e$ such that $E_G(V(T_v)) = \{e, f\}$ holds. □

Lemma 14 *Let $G = (V, E)$ be a 2-edge-connected undirected graph with $|V| \geq 3$, T be a DFS-tree of G rooted at $r \in V$, and P be a degree-two segment in G that does not contain r or any leaf of T. The following are equivalent:*

(i) *P is a MinRS of G;*

(ii) *$G_P - e_P$ is 2-edge-connected; and*

(iii) *$\mathit{CovMult}(e_P) \geq 2$ in G_P.*

Proof: The equivalence between (i) and (ii) is immediate from the definition of MinRSs and the construction of G_P.
(ii) $\iff$ $G_P - e_P$ has no bridge $\iff$ G_P has no 2-edge-cut containing e_P $\iff$ (iii), where the last equivalence follows by Lemma 13. □

Theorem 2 *Given a 2-edge-connected undirected graph $G = (V, E)$ with $n = |V|$ and $m = |E|$, and its DFS-tree T rooted at $r \in V$, we can compute all MinRSs of G consisting of only internal vertices of T that form a degree-two segment in G in $O(n + m)$ time.*

Proof: Let P be a degree-two segment in G that does not contain r or any leaf of T and v denote the vertex in P closest to r in T. By Lemmas 12 and 14, we can determine whether P is a MinRS of G by checking whether $\mathit{CovMult}(\pi(v)v) \geq 2$ in G_P. By Lemma 5, we can compute the cover multiplicities for all tree edges in $O(n + m)$ time. □

4.3 Single Internal Vertex Degree At Least 3 MinRSs

Lemma 15 *Let $G = (V, E)$ be a 2-edge-connected undirected graph with $|V| \geq 3$ and $v \in V$. The graph $G - v$ has a bridge e if and only if there exists $X \subsetneq V \setminus \{v\}$ such that $E_G(X) = \{e\} \cup E_G(v, X)$.*

Proof: ($\Longrightarrow$) Let X be the vertex set of a connected component of $(G - v) - e$, where $E_{G-v}(X) = \{e\}$ holds. Since G is connected, all edges between X and $V \setminus X$ in G must be incident to v. Thus, $E_G(X) = \{e\} \cup E_G(v, X)$ holds.
($\Longleftarrow$) Suppose that there exists $X \subsetneq V \setminus \{v\}$ such that $E_G(X) = \{e\} \cup E_G(v, X)$. In $G - v$,

all edges between X and $V \setminus X$ are removed except for e. Thus, e is a bridge in $G - v$.
$\square$

TODO

4.4 Reuse of DFS-tree and Covering Information

TODO

References

[1] Taishu Ito, Yusuke Sano, Katsuhisa Yamanaka, and Takashi Hirayama. A polynomial delay algorithm for enumerating 2-edge-connected induced subgraphs. *IEICE Transactions on Information and Systems*, E105.D(3):466–473, 2022. `doi:10.1587/transinf.2021FCP0005`.

[2] Takumi Tada and Kazuya Haraguchi. A linear delay algorithm in SD set system and its application to subgraph enumeration. *Journal of Computer and System Sciences*, 152:103637, February 2025. `doi:10.1016/j.jcss.2025.103637`.