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

SHOTA, Kan   HARAGUCHI, Kazuya

December 2, 2025

1 Introduction

2 Preliminaries

Throughout the paper, we assume that a graph is simple and undirected.

For a graph $G = (V, E)$ with a vertex set V and an edge set E, we may abbreviate an edge $\{u, v\} \in E$ into uv (or equivalently vu) for simplicity. For a vertex $v \in V$, we denote by $N_G(v)$ the set of neighbors of v. For $S \subseteq V$, 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 *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 $v \in V$, we denote by T_v the subtree of T rooted at v.

An edge in E is called a *tree edge* if it belongs to 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. If u is an ancestor of v and $u \neq v$, then we say that u is a *proper ancestor* of v, and that v is a *descendant* (resp., 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 it by $\pi(v) = u$.

During the DFS, each vertex is assigned a unique integer called *DFS-index*, which indicates the order of visiting time in the DFS. For $v \in V$, we denote by $dfsId(v)$ the DFS-index of v. Every vertex has a smaller DFS-index than any of its descendants.

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 and the *incoming edge* to v.

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 $Cov(uv) = \{(x, y) \in$

$E \mid (x, y)$ is a back edge covering $uv\}$ the set of back edges that cover uv. We refer to $\mathit{CovMult}(uv) := |\mathit{Cov}(uv)|$ as the *cover multiplicity* of the tree edge uv. Furthermore, we define the *cover-source multiplicity* of uv to be $\mathit{CovSrcMult}(uv) = \big|\{x \in V(T_v) \mid$ There is y s.t. $(x, y) \in \mathit{Cov}(uv)\}\big|$, that is, the number of distinct descendants that serve as tails of back edges covering uv.

For a vertex $x \in V$, every vertex y with a back edge (x, y) lies on the unique $\pi(x)$–r path in T. Among such vertices y, let y^* be the one with the smallest DFS-index, that is, the one closest to the root r in T. We call the back edge xy^* the *longest back edge outgoing from* x and denote it by $\mathit{lbe}(x)$ and y^* by $\mathit{lbeTarget}(x)$.

3 Computation of Cover multiplicities

In this section, we present an $O(n + m)$-time algorithm to compute the cover multiplicity and the cover-source multiplicity for all tree edges in a 2-edge-connected undirected graph $G = (V, E)$ with $n = |V|$ and $m = |E|$.

Since the two multiplicities can be computed in a similar manner, we here describe the algorithm for computing the cover multiplicities; the cover-source multiplicities can be computed with a minor modification as described at the end of this section.

We first perform a DFS on G starting from an arbitrary vertex $r \in V$ and let T be the resulting DFS-tree. The algorithm consists of two phases. In the first phase, we process each back edge to update an auxiliary value called *cover-delta value* associated with its endpoints. In the second phase, we traverse T in a depth-first manner to aggregate these values along the tree and compute the multiplicity for each tree edge (Algorithm 1).

In order to compute the cover multiplicities, we define the *cover-delta value* $\delta(v)$ for each vertex $v \in V$ as follows:

$$\delta(v) := \big|\{(v, y) \mid (v, y) \text{ is a back edge}\}\big| - \big|\{(x, v) \mid (x, v) \text{ is a back edge}\}\big|.$$

Equivalently, if we orient all back edges from descendants to ancestors as described in Section 2 and remove all tree edges, then $\delta(v)$ equals the outdegree minus the indegree of v in the resulting directed graph. The cover-delta values satisfy a fundamental relation with the cover multiplicity, which we show below.

Lemma 1 *For any tree edge (u, v) of G, the cover multiplicity of (u, v) is equal to the sum of the cover-delta values for all vertices in T_v:*

$$\mathit{CovMult}(uv) = \sum_{w \in V(T_v)} \delta(w).$$

Proof: By the definition of δ, every back edge (x, y) contributes $+1$ to $\delta(x)$ and -1 to $\delta(y)$. We examine how such a back edge 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 (x, y) covers uv and contributes $+1$ to the sum. If both x and y belong to $V(T_v)$, then (x, y) does not cover uv and its $+1$ and -1 contributions

Algorithm 1 Computation of Cover Multiplicities by DFS

Input: A 2-edge-connected undirected graph $G = (V, E)$, a DFS-tree T of G rooted at $r \in V$, and the cover-delta values δ for all vertices in V

Output: The cover multiplicities of all tree edges

$\mathit{CovMult}$:= an array from tree edges to integers;
for each child w of the root r in T **do**
    Execute CoverMultiplicity-DFS(w)
end for;
output $\mathit{CovMult}$

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

cancel within T_v, yielding a total contribution of 0. If neither x nor y is in T_v, then (x, y) does not cover uv 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 back edges that contribute a nonzero amount are those that cover uv, and each such edge contributes exactly +1. □

In order to compute the cover-delta values for all vertices in V, we can process each back edge (x, y) of G and increment $\delta(x)$ by 1 and decrement $\delta(y)$ by 1. This can be done in $O(n+m)$ time by performing a DFS on G to obtain its DFS-tree T and then processing all back edges of G.

The second phase computes the cover multiplicities for all tree edges by aggregating the cover-delta values along the DFS-tree T. The details are shown in Algorithm 1.

Lemma 2 *Given a 2-edge-connected undirected graph $G = (V, E)$ with $n = |V|$ and $m = |E|$, its DFS-tree T rooted at $r \in V$, and the cover-delta values δ for all vertices in V, Algorithm 1 computes the cover multiplicities for all tree edges in $O(n)$ time.*

Proof: By Lemma 1, it suffices to show that Algorithm 1 correctly computes the sum of the cover-delta values for all vertices in T_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 returns $\mathit{CovMult}(uv) = \delta(v)$.

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

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

that is the sum of the cover-delta values for all vertices in T_v.

We visit each vertex exactly once, and consider all tree edges exactly once, which means we can correctly compute the cover multiplicities for all tree edges. Since we traverse only tree edges in T, we can execute the algorithm in $O(n)$ time. □

To compute the cover-source multiplicities, we modify the definition of the cover-delta value so that it accounts only for the longest outgoing back edges. We define the *cover-source-delta* value $\delta_{\mathrm{src}}(v)$ to be

$$\delta_{\mathrm{src}}(v) := \begin{Bmatrix} 1 & \text{if } \mathit{lbe}(v) \text{ exists} \\ 0 & \text{otherwise} \end{Bmatrix} - \bigl|\{x \in V \mid \mathit{lbe}(x) = (x, v)\}\bigr|.$$

Equivalently, if we orient only the longest back edges from descendants to ancestors and remove all the other edges, then $\delta_{\mathrm{src}}(v)$ equals the outdegree minus the indegree of v in the resulting directed graph. Then, by a similar argument to Lemma 1, we can show that

$$\mathit{CovSrcMult}(uv) = \sum_{w \in V(T_v)} \delta_{\mathrm{src}}(w).$$

Thus, by replacing δ with δ_{src} in Algorithm 1, we can compute the cover-source multiplicities for all tree edges.

Therefore, we obtain the following theorem that summarises the results in this section.

Theorem 1 *Given a 2-edge-connected undirected graph $G = (V, E)$ with $n = |V|$ and $m = |E|$, we can compute the cover multiplicity and the cover-source multiplicity for all tree edges in G in $O(n + m)$ time.*

PROOF: We first perform a DFS on G to obtain its DFS-tree T rooted at an arbitrary vertex $r \in V$ in $O(n + m)$ time. Next, we process all back edges of T to compute the cover-delta values δ or cover-source-delta values δ_{src} for all vertices in V in $O(n + m)$ time. Finally, we execute Algorithm 1 with each tree edge (r, v) incident to the root r to compute the cover multiplicities for all tree edges in T. By Lemma 2, this step takes $O(n)$ time. Thus, the total time complexity is $O(n + m)$. □

4 Mathematical Properties of Minimal Removable Sets

4.1 DFS-Tree Leaf MinRSs

In this section, 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 3 *Let $G = (V, E)$ be a 2-edge-connected undirected graph and T be a DFS-tree of G rooted at $r \in V$. Any leaf is not an articulation point of G.*

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, $T - v$ is connected. Thus, $G - v$ is also connected, which means that v is not an articulation point of G. □

Lemma 4 *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 x be a leaf of T that has back edges and let $y := lbeTarget(x)$. Then, $\{x\}$ is a MinRS of G if and only if, for any tree edge (u, v) on the y–$\pi(x)$ path in T, there exists a back edge outgoing from some vertex in $T_v - x$ that covers (u, v).*

PROOF: ($\Longrightarrow$) We show by contraposition. Suppose that there exists a tree edge (u, v) on the y–$\pi(x)$ path in T that is not covered by any back edge outgoing from a vertex other than x. Any vertex in $T_v - x$ cannot have a back edge to a proper ancestor of u in $G - x$, which means that uv is a bridge in $G - x$. Thus, $G - x$ is not 2-edge-connected.

($\Longleftarrow$) We suffice to show that $G - x$ is 2-edge-connected. By Lemma 3, $G - x$ is connected. We show that there is no bridge in $G - x$. Every back edge (x', y') with $x', y' \neq x$ is not a bridge in $G - x$, since $x'y'$ and the y'–x' path in T form a cycle in $G - x$. Next, we consider tree edges in $G - x$. For any tree edge not on the y–$\pi(x)$-path in T, no covering back edge disappears by removing x from G, which means that such a tree edge is not a bridge in $G - x$. For any tree edge (u, v) on the y–$\pi(x)$-path in T, by the assumption, there exists a back edge outgoing from a vertex in $T_v - x$ that covers (u, v), which means that such a tree edge is not a bridge in $G - x$. Thus, there is no bridge in $G - x$, and hence, $G - x$ is 2-edge-connected. □

Here, we introduce the *source-exclusive coverage set* $SrcExc_v$ of v for each vertex $v \in V$, defined to be the set of tree edge e in T such that $CovSrcMult(e) = 1$ and e is covered by back edges outgoing from v. Or equivalently, if $e \in SrcExc_v$, then e is covered by back edges outgoing from v, and no back edge outgoing from any other vertex covers e. Any tree edge in $SrcExc_v$ must be on the $lbeTarget(v)$–$\pi(v)$ path in T.

By Lemma 4, $\{v\}$ is a MinRS of G if and only if $SrcExc_v = \emptyset$. Thus, in order to find all MinRSs consisting of only leaves of T, it suffices to compute $SrcExc_v$ for each leaf v of T. This can be done by the Algorithm 2 that traverses T in a depth-first manner and collects tree edges with cover-source multiplicity 1 covered by back edges outgoing from each leaf.

Lemma 5 *Given a 2-edge-connected undirected graph $G = (V, E)$ with $n = |V|$ and $m = |E|$, its DFS-tree T rooted at $r \in V$, and the cover-source multiplicities $CovSrcMult$ for all tree edges in T, Algorithm 2 computes $SrcExc_v$ for each leaf v of T in $O(n)$ time.*

Algorithm 2 Computation of source-exclusive coverage set for Each Leaf

Input: A 2-edge-connected undirected graph $G = (V, E)$, DFS-tree T of G rooted at $r \in V$, and the cover-source multiplicities *CovSrcMult* for all tree edges in T

Output: The source-exclusive coverage sets SrcExc_v for all leaves v of T

Initialise $\mathit{SrcExc}_v := \emptyset$ for all leaves v of T;
for each child w of the root r in T **do**
    Execute TreeEdgeToBackEdge(w)
end for;
output SrcExc_v for all leaves v of T

procedure TreeEdgeToBackEdge(v)
    $l :=$ Nil;
    if v is a leaf **then**
        $l := v$
    end if;
    for each child w of v in T **do**
        Execute TreeEdgeToBackEdge(w);
        Let l' be the returned leaf;
        if $l' \neq$ Nil and [$l =$ Nil or $\mathit{dfsId}(\mathit{lbeTarget}(l')) < \mathit{dfsId}(\mathit{lbeTarget}(l))$] **then**
            $l := l'$
        end if
    end for;
    $\mathit{SrcExc}_l := \mathit{SrcExc}_l \cup \{uv\}$ **if** $\mathit{CovSrcMult}(uv) = 1$; $\triangleright$ l is not Nil if this line is executed
    return l
end procedure

Proof: TODO □

Theorem 2 *Given a 2-edge-connected undirected graph $G = (V, E)$ with $n = |V|$ and $m = |E|$, we can compute all MinRSs consisting of only leaves of a DFS-tree in $O(n + m)$ time.*

Proof: As the given graph is 2-edge-connected, each tree edge is covered by at least one back edge. By Lemma 4, $\{v\}$ is a MinRS of G if and only if for any tree edge is covered by back edges outgoing from vertices other than v, which means that $\mathit{CovSrcMult}(e) \geq 2$. Thus, we can check whether $\{v\}$ is a MinRS of G by checking whether BE_v is empty or not. By Theorem 1, we can compute the cover-source multiplicities for all tree edges in $O(n + m)$ time. By Lemma 5, we can compute the sets BE_v for all leaves v of T in $O(n)$ time. Thus, we can compute all MinRSs consisting of only leaves of T in $O(n + m)$ time. □

4.2 Maximal 2-deg Path MiRSs

TODO

4.3 Single Vertex Degree at Least 3 MinRSs

TODO

4.4 Reuse of DFS-tree and Covering Information

TODO