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

SHOTA, Kan HARAGUCHI, Kazuya

23rd 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. 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 $G - \{v\}$.

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 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. 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 $\pi(v) = u$.

During the DFS, each vertex is assigned a unique integer called *DFS-index*, which records the order in which the DFS visits the vertices. For $v \in V$, we denote its DFS-index by $\mathit{dfsId}(v)$. 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 $\mathit{Cov}(uv) = \{(x, y) \in E \mid (x, y) \text{ is a back edge covering } uv\}$ the set of back edges that cover uv. We refer to $\mathit{CovMult}(uv) \triangleq |\mathit{Cov}(uv)|$ as the *cover multiplicity* of the tree edge uv. Furthermore, we define the *cover-source multiplicity* of uv to be $\mathit{CovSrcMult}(uv) \triangleq \big|\{x \in V(T_v) \mid \text{There is } y \text{ s.t. } (x, y) \in \mathit{Cov}(uv)\}\big|$, that is, the number of distinct descendants of v that serve as tails of back edges covering uv.

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

$$\mathit{CovDesc}_i(uv) \triangleq \sum_{(x,y)\in \mathit{Cov}(uv)} (id(x))^i, \qquad \mathit{CovAnc}_i(uv) \triangleq \sum_{(x,y)\in \mathit{Cov}(uv)} (id(y))^i.$$

We refer to these quantities as *weighted cover information.*

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^* denote 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 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 weighted cover information 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 weighted cover information.

Lemma 1 *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 $\mathit{Cov}(uv) = \{(x_1, y_1), \ldots, (x_q, y_q)\}$ be the set of back edges covering uv, where $q = \mathit{CovMult}(uv)$. Then, the following two conditions hold:*

- *$x_1 = x_2 = \cdots = x_q$ if and only if $(\mathit{CovDesc}_1(uv))^2 = q \cdot \mathit{CovDesc}_2(uv)$; and*
- *$y_1 = y_2 = \cdots = y_q$ if and only if $(\mathit{CovAnc}_1(uv))^2 = q \cdot \mathit{CovAnc}_2(uv)$.*

Proof: We prove the descendant-side condition.

$$\begin{aligned}
&q \cdot \mathit{CovDesc}_2(uv) - (\mathit{CovDesc}_1(uv))^2 \\
&= q \cdot \sum_{i=1}^{q} (id(x_i))^2 - \left(\sum_{i=1}^{q} id(x_i)\right)^2 \\
&= \sum_{i=1}^{q} (q-1)\,(id(x_i))^2 - \sum_{i=1}^{q-1}\sum_{j=i+1}^{q} 2id(x_i)\cdot id(x_j) \\
&= \sum_{i=1}^{q-1}\sum_{j=i+1}^{q} (id(x_i) - id(x_j))^2\,;
\end{aligned}$$

the last equality holds by rearranging the terms. Since each term in the last summation is non-negative, the entire summation is equal to zero if and only if $id(x_i) = id(x_j)$ for all $i, j \in \{1, \dots, q\}$, which is equivalent to $x_1 = x_2 = \cdots = x_q$. The ancestor-side condition can be proved similarly. $\square$

3.1 Difference-Based Aggregation on a Rooted Tree

Let T be a rooted tree. Each tree 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 tree 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 tree edges. The goal is to compute the final value stored in each tree 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:

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

Lemma 2 *Let T be a rooted tree and each tree 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 tree 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

Algorithm 1 Difference-Based Aggregation on a Rooted Tree

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

Output: The final values for all tree edges

```
1:  Initialise $\Delta(v) := 0$ for all $v \in V(T)$ and $value(e) := A(e)$ for all $e \in E(T)$;
2:  for each update operation $(x, y, \alpha)$ do
3:      $\Delta(x) := \Delta(x) + \alpha$; $\Delta(y) := \Delta(y) - \alpha$
4:  end for;
5:  for each child $w$ of the root $r$ in $T$ do
6:      Execute AGGREGATE-DFS($w$)
7:  end for;
8:  output $value(e)$ for all tree edges $e$ in $E(T)$
9:
10: procedure AGGREGATE-DFS($v$)
11:     $value(\pi(v)v) := value(\pi(v)v) + \Delta(v)$;
12:     for each child $w$ of $v$ in $T$ do
13:         Execute AGGREGATE-DFS($w$);
14:         $value(\pi(v)v) := value(\pi(v)v) + value(vw)$
15:     end for
16: end procedure
```

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 $+\alpha$. □

Lemma 3 *Given a rooted tree T with n vertices and a sequence of q update operations, Algorithm 1 computes the final values for all tree edges in $O(q + n)$ time.*

PROOF: By Lemma 2, it suffices to show that Algorithm 1 correctly computes $A(\pi(v)v) + \sum_{w \in V(T_v)} \Delta(w)$. for each tree 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 $value(\pi(v)v) = A(\pi(v)v) + \Delta(v)$.

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 $O(1)$ time, the total time for all q update operations is $O(q)$. We visit each vertex exactly once, and consider all tree edges exactly once, which means we can correctly compute the final values for all tree edges. Since we traverse only tree edges in T, we can execute the aggregation in $O(n)$ time. $\square$

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 4 *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 $\mathit{CovMult}(uv)$;*
- *Cover-source multiplicity $\mathit{CovSrcMult}(uv)$;*
- *Weighted cover information $\mathit{CovDesc}_1(uv)$ and $\mathit{CovDesc}_2(uv)$;*
- *Weighted cover information $\mathit{CovAnc}_1(uv)$ and $\mathit{CovAnc}_2(uv)$.*

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 $\mathit{CovSrcMult}(uv)$, set $\alpha := 1$ if $(x, y) = \mathit{lbe}(x)$; otherwise, set $\alpha := 0$;
- For $\mathit{CovDesc}_i(uv)$, set $\alpha := \mathit{id}(x)^i$ for $i \in \{1, 2\}$; and
- For $\mathit{CovAnc}_i(uv)$, set $\alpha := \mathit{id}(y)^i$ 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 3, we can compute the final values for all tree edges in $O(n + m)$ time. $\square$

4 Mathematical Properties and Computation 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 5 *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 $r \in V$. Any leaf of T 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. $\square$

Lemma 6 *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 and $y := \mathit{lbeTarget}(x)$. 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.

($\Longleftarrow$) It suffices to show that $G - x$ is 2-edge-connected. By Lemma 5, $G - x$ is connected. We show that there is no bridge in $G - x$. Every back edge $(x', y') \in E \setminus E(T)$ 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. $\square$

Here, we introduce the *source-exclusive coverage set* SrcExc_v of each vertex $v \in V$, defined to be the set of tree edge e in T such that $\mathit{CovSrcMult}(e) = 1$ and e is covered by only back edges outgoing from v. Or equivalently, if $e \in \mathit{SrcExc}_v$, 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 $\mathit{lbeTarget}(v)$–$\pi(v)$ path in T.

By Lemma 6, $\{x\}$ is a MinRS of G if and only if $\mathit{SrcExc}_x = \{\pi(x)x\}$. 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. Before describing the algorithm, there are some lemmas to show its correctness.

Lemma 7 *Let $G = (V, E)$ be a 2-edge-connected undirected graph and T be a DFS-tree of G rooted at $r \in V$. Let l be a leaf of T and $v \neq r$ be an ancestor of l in T. The tree edge $\pi(v)v$ is covered by the longest back edge $lbe(l)$ outgoing from l if and only if $dfsId(lbeTarget(l)) < dfsId(v)$.*

PROOF: Let $y := lbeTarget(l)$. Since both y and v are on the unique $\pi(l)$–r path in T, y is an ancestor or a descendant of v. We have $dfsId(y) < dfsId(v)$ if and only if y is a proper ancestor of v. By the definition of coverage, $\pi(v)v$ is covered by $lbe(l)$ if and only if y is a proper ancestor of v. □

Lemma 8 *Let $G = (V, E)$ be a 2-edge-connected undirected graph and T be a DFS-tree of G rooted at $r \in V$. Let l, l' be distinct leaves of T and $v \neq r$ be a common ancestor of l and l' in T. We assume that $dfsId(lbeTarget(l)) \geq dfsId(lbeTarget(l'))$. If a tree edge $\pi(v)v$ is covered by the longest back edge $lbe(l)$ outgoing from l, then $\pi(v)v$ is also covered by the longest back edge $lbe(l')$ outgoing from l'.*

PROOF: By Lemma 7, since $\pi(v)v$ is covered by $lbe(l)$, we have $dfsId(lbeTarget(l)) < dfsId(v)$. By the assumption, $dfsId(lbeTarget(l')) \leq dfsId(lbeTarget(l)) < dfsId(v)$ holds. By Lemma 7 again, $\pi(v)v$ is also covered by $lbe(l')$. □

We are now ready to describe Algorithm 2 that computes $SrcExc_v$ for each leaf v of T.

Lemma 9 *Given a 2-edge-connected undirected graph $G = (V, E)$ with $n = |V| \geq 3$ 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.*

PROOF: (Correctness) We first show that the case with v being a leaf. In this case, since there is no child of v in T, the algorithm returns $l = v$. Moreover, $\pi(l)l$ is clearly covered by $lbe(l)$ and no longest back edge outgoing from other leaf covers $\pi(l)l$. This means that, if $CovSrcMult(\pi(l)l) = 1$, then $\pi(l)l \in SrcExc_l$ holds.

Next, we consider the case other than the leaf case. We assume that, for any child w of v in T, TREEEDGETOBACKEDGE(w) correctly computes a leaf l' such that every tree edge in $SrcExc_{l'}$ is covered by $lbe(l')$ and no longest back edge outgoing from other leaf covers such tree edges. By Lemma 8, we need to consider only the leaf l' with the smallest $dfsId(lbeTarget(l'))$ among such leaves l' returned from children of v. The algorithm correctly selects such a leaf l (line 12). If $dfsId(lbeTarget(l)) \geq dfsId(v)$, then, by Lemma 7, no longest back edge outgoing from l covers $\pi(v)v$, and hence, we set $l :=$ NIL

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

```
1: Initialise $\mathit{SrcExc}_v := \emptyset$ for all leaves $v$ of $T$;
2: for each child $w$ of the root $r$ in $T$ do
3:     Execute TreeEdgeToBackEdge($w$)
4: end for;
5: output $\mathit{SrcExc}_v$ for all leaves $v$ of $T$
6:
7: procedure TreeEdgeToBackEdge($v$)
8:     $l := \begin{cases} v & \text{if } v \text{ is a leaf} \\ \textsc{Nil} & \text{otherwise} \end{cases}$;
9:     for each child $w$ of $v$ in $T$ do
10:        Execute TreeEdgeToBackEdge($w$);
11:        Let $l'$ be the returned leaf;
12:        if $l' \neq \textsc{Nil}$ and $[l = \textsc{Nil}$ or $\mathit{dfsId}(\mathit{lbeTarget}(l')) < \mathit{dfsId}(\mathit{lbeTarget}(l))]$ then
13:            $l := l'$
14:        end if
15:    end for;
16:    if $l \neq \textsc{Nil}$ and $\mathit{dfsId}(\mathit{lbeTarget}(l)) \geq \mathit{dfsId}(v)$ then
17:        $l := \textsc{Nil}$
18:    end if;
19:    $\mathit{SrcExc}_l := \mathit{SrcExc}_l \cup \{\pi(v)v\}$ if $\mathit{CovSrcMult}(\pi(v)v) = 1$ and $l \neq \textsc{Nil}$;
20:    return $l$
21: end procedure
```

(line 16). Then, if $l \neq \textsc{Nil}$, then the tree edge $\pi(v)v$ is covered by $\mathit{lbe}(l)$. Therefore, if $\mathit{CovSrcMult}(\pi(v)v) = 1$ and $l \neq \textsc{Nil}$, then we can insert $\pi(v)v$ into SrcExc_l.

(Time complexity) We visit each vertex exactly once, and consider all tree edges exactly once, which means we can compute SrcExc_v for all leaves v of T. Since we traverse only tree edges in T, we can execute the algorithm in $O(n)$ time. □

Theorem 1 *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 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. By Lemma 6, $\{v\}$ 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 SrcExc_v is equal to $\{\pi(v)v\}$ ot not. By Lemma 4, we can compute the cover-source multiplicities for all tree edges in $O(n+m)$ time. By Lemma 9, we can compute the sets SrcExc_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 Link MinRSs

TODO

4.3 Single Vertex Degree at Least 3 MinRSs

TODO

4.4 Reuse of DFS-tree and Covering Information

TODO