Near-Linear Preprocessing for Reachability After Two Vertex Failures in DAGs and Reducible Flowgraphs

Idris Ali Shaik
Independent researcher (unaffiliated)
shaikidris@apache.org

Version 1.4.1, 29 November 2026.

hexagon:2611.01364v1

Abstract

We construct a single-source reachability oracle for directed acyclic graphs and reducible flowgraphs. After preprocessing, a query specifies a target and up to two failed vertices and asks whether a path to the target survives. For n vertices and m arcs, the deterministic construction takes $O((n+m)\log(n+2))$ time and $O(n+m)$ working words. The oracle retains $O(n)$ words and answers in $O(1)$ worst-case time. The construction groups vertices by their immediate dominator. Within each group, the pairs whose deletion disconnects another vertex form a forest. An identity for exits from dominator subtrees reduces a query to a constant number of tree-path extrema. The same preprocessing bound gives a subgraph with at most four incoming arcs per vertex preserving all two-failure answers. When the target is fixed before preprocessing, we obtain the same space and query bounds for three arbitrary vertex failures. A Lean 4 formalization of Theorem 1.1, including its resource bounds, is registered with Palomar.

1. The oracle

A single vertex a disconnects a reachable target t from a source r exactly when every r-to-t path visits a. The dominator tree stores that relation for every target. Two vertices can disconnect t even if neither lies on every path. For this problem Choudhary gives a linear-space, constant-query-time oracle for general directed graphs with $O(mn)$ preprocessing [Cho16, Cho17]. We ask whether one can retain these query and space bounds while building the oracle in near-linear time when the graph is acyclic. Theorem 1.1 does so and also applies to reducible flowgraphs.

The graph is given by adjacency lists with $n \geq 1$ vertices, m arc occurrences and a specified source r. Parallel arcs are allowed. Write $L = \lceil \log_2(n+2) \rceil$. For a set F of failed vertices let $R(F)$ be the vertices reachable from r in $G - F$. A path may have length zero, and $R(F) = \varnothing$ if $r \in F$. Each query starts from the original graph. A vertex dominates another if it lies on every path from the source to that vertex. A flowgraph is *reducible* exactly when removing the arcs whose heads dominate their tails leaves a DAG [HU74]. The cyclic case below assumes reducibility on the vertices reachable from r. Costs count deterministic operations on words of $\Theta(\log(n+m+2))$ bits, or $\Theta(\log(S+2))$ bits for an explicit batch of size S.

Theorem 1.1 (two-failure reachability oracle). For a DAG or a reducible flowgraph with fixed source r, one can construct an index in $O((n+m)L)$ time and $O(n+m)$ working words. The index occupies $O(n)$ words. Given any original target t and any set F of at most two original vertices after construction, it decides whether $t \in R(F)$ in $O(1)$ worst-case time.

The index retains the original tail of each arc leaving a dominator subtree. A failure inside the subtree may disable some exits and leave others reachable. All the tails disabled by that failure lie in one interval of dominator-tree preorder. It therefore suffices to know the smallest and largest ranks of the tails whose exits lead to the target. Sections 2 and 3 identify the separating pairs. Theorem 4.1 expresses the exits at one pair in terms of the exits at its child. Section 5 uses this identity to recover the two extrema in constant time.

Corollary 1.2 (listed target sets). Suppose q queries specify failure sets F_i of size at most two and explicit target lists T_i. Put $M = \sum_i |T_i|$ and $S = n + m + q + M$. Computing one Boolean $[T_i \cap R(F_i) \neq \varnothing]$ per list takes $O((n+m)L + q + M)$ total time and $O(S)$ working words, including construction. A single list given later takes $O(1 + |T_i|)$ time.

Proof. Build the oracle once. Scan each list until a target survives or the list is exhausted. This makes at most M constant-time probes. Reading the failure sets and writing the q answers costs $O(q)$, including empty

lists. □

Corollary 1.3 (three failures with a fixed target). Fix t as well as r before construction on a DAG or a reducible flowgraph. An index built in $O((n+m)L)$ time and $O(n+m)$ working words retains $O(n)$ words and decides whether $t \in R(F)$ in $O(1)$ worst-case time for any later set F of at most three vertices. Two targets fixed in advance can both be answered within the same bounds.

The proof in Section 6 orders three internal failures along the DAG. A path survives if it can bypass the last failure from a tail reachable after the first two. With a fixed target, we can prepare the candidate tails for every position of the last failure. Corollary 6.2 selects a subgraph that preserves all two-failure answers using at most $4(n-1)$ original arcs.

Single-source oracle	Failures and target	Build time	Retained words	Query time
Choudhary, general digraphs [Cho17, Theorem 3.2]	Up to two vertices, any later target	$O(mn)$	$O(n)$	$O(1)$
Theorem 1.1, DAGs and reducible flowgraphs	Up to two vertices, any later target	$O((n+m)L)$	$O(n)$	$O(1)$
Corollary 1.3, same graph classes	Up to three vertices, fixed target	$O((n+m)L)$	$O(n)$	$O(1)$
Proposition 5.3, Lean-verified implementation on DAGs	Up to two vertices, any later target	$O((n+m)L^2)$	$O(n)$	$O(L)$

Related work. Choudhary tests for detours around failed vertices on two independent spanning trees [Cho16, Theorem 13]. Our preprocessing uses topological order, first to find the separating pairs and then to compute the exits between them. Teslenko and Dubrova construct a linear-size representation of all double dominators of a given vertex in linear time [TD15]. In our path orientation this fixes the target. Teslenko describes the quadratic cost of building separate chain representations for all vertices [Tes08, §3.9]. Teslenko, Dubrova and Tenhunen combine the prepared dominator chains of k supplied circuit vertices in $O(k \log n)$ time [TDT06, §§5–6]. Theorem 1.1 constructs one index from the input graph and accepts the target at query time.

The graph on siblings in the dominator tree is Tarjan's derived graph [GT15, §3]. For binary decision diagrams, Karplus represents two-cuts by child pairs linked to the next cut and gives a quadratic-time construction [Kar88, §§2.3–2.4]. Here the sibling graph retains original arc tails, the cut forest permits arbitrary indegrees, and exit information passes between adjacent cuts. Proposition 5.3 states the bounds of the DAG implementation verified in Lean 4 [ShaL26].

2. Where a surviving path can leave a block

Discard vertices unreachable from r, retaining a translation for original identifiers. They can never become reachable through deletions. Also discard them from failure sets. On the retained graph, *a dominates v* if every r-to-v path contains a. The dominators of v are its ancestors in a tree $\mathcal{D}$ rooted at r. Write $D(a)$ for the subtree rooted at a, including a. Thus deleting a loses exactly $D(a)$. We can find $\mathcal{D}$ within $O((n+m)L)$ time and linear working space by the simple Lengauer–Tarjan algorithm [LT79].

Removing cycles in a reducible flowgraph. After finding $\mathcal{D}$ on the reachable graph, remove every arc $x \to w$ for which w dominates x, including self-loops. A simple surviving path could use such an arc only

by visiting w twice, since its prefix to x already contains w. Every reachable target admits a simple path. Deleting all these arcs together therefore preserves $R(F)$ for *every* vertex-failure set F. The Hecht–Ullman characterization [HU74] says the remaining graph is acyclic exactly when the reachable flowgraph is reducible. Dominator preorder intervals test every arc in one scan. Ordinary dominators are preserved as well, since the reachable sets after single deletions agree. In the sequel G denotes this reachable DAG.

For a vertex d of $\mathcal{D}$ with children, call each child subtree $D(u)$ a *block* of d. Define H_d to have root d and one vertex for each child. An original arc $x \to y$ is assigned to $d = \mathrm{idom}(y)$, the parent of y in $\mathcal{D}$. If $x = d$, insert $d \to y$. Otherwise insert $u \to y$, where u is the child above x. In both cases label the inserted arc with its original tail x. Two arcs with the same quotient endpoints may have different labels.

Lemma 2.1 (sibling quotients). Each H_d is a rooted DAG in which deleting any nonroot vertex loses only that vertex. Paths through the blocks project to paths in H_d, and a quotient path lifts through any chosen sequence of its labelled arcs. Across all d there are at most $2n$ vertex copies and one quotient arc per retained original arc.

Proof. Any arc into a strict descendant of u from outside $D(u)$ would allow a source path to that descendant avoiding u. Thus entry from outside a block is at its root. Every vertex of the block has an internal path from its root. Indeed, a source path to it passes through the root and could only reenter at that root, which acyclicity prevents. The same reasoning applies to $D(d)$.

Every strict dominator of y dominates the tail x of an entering arc, since any source path to x extends to y. The tail lies outside $D(y)$, since otherwise y would precede x, contradicting the arc $x \to y$. The quotient arc is therefore well defined, and its tail precedes its head in original topological order. Project an original path segment starting at d and ending at a child of d by replacing each visit to a block by its root. Conversely, upon entering a block, follow an internal path to the original tail of the next specified arc. This lifts the quotient path, including the choice among parallel exits.

For distinct children u, y, the vertex u does not dominate y. A path to y avoiding u avoids its whole block. The suffix from d projects to a path avoiding u in H_d. Hence every nonroot vertex there has immediate dominator d. Each child occurs once as a child and at most once as a quotient root. Arc assignment uses the unique parent of its head. □

If the source fails, or if an original target was discarded, the answer is immediate. A single retained failure is handled by a dominator ancestry test. For two comparable failures the higher one suffices, since every source path visiting the lower one also visits the higher one. Suppose henceforth a, b are incomparable in $\mathcal{D}$. Let d be their lowest common ancestor and u, v the distinct children of d above them. Write $B = \{u, v\}$. For a pair B in H_d, let $Z(B)$ be all vertices lost on deleting B, including B, and let $J(B) = Z(B) \setminus B$ be its *proper losses*.

For $y \in J(B)$ and $s \in B$, let $X_s^B(y)$ be the set of original tails on quotient arcs $s \to w$ such that w reaches y in $H_d - B$. A direct arc to y qualifies. Both endpoint sets are nonempty. Deleting the other endpoint alone preserves y, and the surviving path must visit s. Its last exit from s supplies a tail. Number the vertices of the original dominator tree in preorder. The ranks in $D(a)$ form the half-open interval $I(a) = [\mathrm{tin}(a), \mathrm{tout}(a))$, where $\mathrm{tout}(a)$ is the first rank after the subtree. For a nonempty set X of original vertices write

$$\sigma(X) = \big[\min_{x \in X} \mathrm{tin}(x),\ \max_{x \in X} \mathrm{tin}(x)\big], \qquad \sigma_s^B(y) = \sigma(X_s^B(y)).$$

Set $\sigma(\varnothing) = \varnothing$.

Lemma 2.2 (last-exit test). Every target outside $D(d)$, and d itself, survives. A target in $D(u)$ survives precisely when a does not dominate it, with the symmetric rule for $D(v)$. For a target in a third block $D(y)$, it survives if $y \notin J(B)$. If $y \in J(B)$, it survives precisely when

$$\sigma_u^B(y) \not\subseteq I(a) \quad \text{or} \quad \sigma_v^B(y) \not\subseteq I(b).$$

Proof. Outside $D(d)$ there is a path avoiding d, so it avoids both failures. The two failures follow d in topological order. To reach a target in $D(u)$ that a does not dominate, first reach u avoiding b. This prefix

avoids a, a strict descendant of u in this case. Take the internal suffix of a path to the target avoiding a. It avoids b because it stays in $D(u)$. If $a = u$, every target of this block is lost. The same argument applies in $D(v)$.

A third block has no internal failure, so reaching y suffices. A quotient path avoiding B lifts by Lemma 2.1. Otherwise, consider a surviving physical path and its last exit from either of the two failure blocks. Its tail is reachable, and the remaining quotient path avoids B. Conversely, any such reachable tail extends to y along the labelled arc and its continuation through failure-free blocks. The preceding within-block test determines which tails are reachable. For nonempty X, all its tails lie in $D(a)$ precisely when its smallest rank is at least $\mathrm{tin}(a)$ and its largest rank is strictly below $\mathrm{tout}(a)$. Applying these two comparisons at each endpoint gives the displayed test. □

3. A forest of separating pairs

Fix one sibling quotient H with root ρ. Its dominator tree is flat, which means every nonroot vertex is a child of ρ. We use B only for pairs of distinct nonroot vertices. Many pairs have no proper loss. Those with a proper loss can all be found in one topological sweep.

Lemma 3.1 (forest of proper losses). There is a rooted forest with one node for each pair B with $J(B) \neq \varnothing$, and a label $c(y)$ for each vertex of H. Adjoin an artificial root 0 above all forest roots, allowing $c(y) = 0$. For every real pair B, $y \in J(B)$ precisely when B is an ancestor of $c(y)$, with ancestry inclusive. A real child P of a real parent B satisfies $P \subseteq Z(B)$ and $J(P) \subseteq J(B)$. Every vertex in $P \setminus B$ follows both endpoints of B in topological order. There are at most $|V(H)|$ real forest nodes.

Proof. A proper loss of a pair follows both endpoints. Before its later endpoint, that deletion has no effect and the earlier endpoint alone loses only itself. Consider the first proper loss. Every predecessor is already lost, so each belongs to the pair. There must be two distinct predecessors, since a unique nonroot predecessor would by itself disconnect the vertex. Thus the two predecessors are precisely the failed pair. Each pair with a proper loss has such a first vertex, and the vertex determines the pair.

Now process H in topological order. At ρ and at vertices with an arc directly from ρ, use label 0. Assume the statement about ancestor labels holds for earlier vertices. Every earlier real forest edge from B to P was installed with $P \subseteq Z(B)$. This implies $Z(P) \subseteq Z(B)$ even for future vertices. Indeed, a path surviving B cannot visit an unreachable vertex, so avoids P.

At another vertex y, let m_y be the latest distinct predecessor. A pair that properly loses y must lose all its predecessors. If that pair omits m_y, it properly loses m_y and occurs on the ancestor path of $c(m_y)$. On that path the real pairs losing every predecessor form a prefix starting nearest the artificial root, by the nesting of the Z sets. Let B_* be the deepest such pair, or 0 for an empty prefix. Conversely every real pair in this prefix properly loses y, since its endpoints already precede m_y.

If a pair contains m_y, each other predecessor can be lost only by being its other endpoint. It precedes m_y and a singleton deletion disconnects only itself. Since y has no direct source arc, it has at least two distinct predecessors. The only additional pair is therefore $P = \{p, m_y\}$ when these are exactly its predecessors. Install P immediately below B_*, and set $c(y) = P$. If this key already occurs, reuse the node. Its parent is determined by the labels and raw memberships at p, m_y, fixed when those vertices were processed. If there is no additional pair, use $c(y) = B_*$. Whenever B_* is real, both endpoints of the new pair belong to $Z(B_*)$, as required for the induction. A pair first encountered at this step cannot acquire an earlier proper loss by the first-vertex argument.

The proper-loss sets are now inverse images under c of forest subtrees. This gives $J(P) \subseteq J(B)$. A fresh endpoint in $P \setminus B$ lies in $Z(B) \setminus B$, so the initial topological observation places it after both old endpoints. □

Lemma 3.2 (constructing the forest). On a flat quotient with n_H vertices and m_H labelled arcs, the labels $c(y)$ and the pair forest can be built in $O((n_H + m_H)L)$ time and $O(n_H + m_H)$ working words. The labels, parents, depths, one ancestor pointer per node and endpoint pairs occupy $O(n_H)$ words.

Proof. In each incoming row gather the distinct predecessors with a timestamp array and record the latest one, m_y. Keep the labelled arcs for Section 5. For a row with exactly two distinct nonroot predecessors and no root arc, form the key consisting of their ordered ranks. Sort the at most n_H candidate keys in $O(n_H L)$ time and link equal keys to a common array cell. The first row processed with a given key fills the cell with its new pair node. Later rows reuse it.

We need ancestor queries while the forest is growing. Use Myers's skew-binary stack [Mye83, §§3–4]. Each node has a depth, a parent and a jump pointer j, with $j(0) = 0$. To insert a child v of p, put $a = j(p)$ and $b = j(a)$. If the jumps from p to a and from a to b have equal positive lengths, set $j(v) = b$. Otherwise set $j(v) = p$.

The list of ancestors starting at v, up to but excluding 0, consists of blocks of sizes $2^k - 1$, $k \geq 1$. Sizes increase from the front, with equality allowed only between the first two. A jump skips the leading block. Insertion either adds a singleton or joins the new head and the two equal leading blocks. This proves the invariant and shows that jump length depends only on depth. To find an ancestor at a given depth, jump whenever the jump stays at or below that depth and otherwise take a parent step. There are $O(L)$ whole blocks before the answer. In the block containing the answer a parent step splits the block into two equal smaller blocks, of which at most one is skipped. Thus there are $O(L)$ moves. The same argument applies to finding a boundary defined by a monotone predicate on the ancestor path, since each proposed landing can be tested.

The lowest common ancestor (LCA) of two online nodes also takes $O(L)$ moves. Equalize their depths. While the nodes differ, jump both when their jump endpoints differ and otherwise take both parents. At equal depths the jumps have equal lengths, and their endpoints differ exactly when the landing is strictly below the LCA. Thus the traversal searches for a single depth boundary, with the same block bound.

For a head with no root arc and latest predecessor m_y, consider the path from 0 to $C = c(m_y)$. For a quotient vertex z, let $r_C(z)$ be the depth of the last real node on this path whose lost set contains z, or zero if none does. By Lemma 3.1 the qualifying nodes form a prefix. Every real node through $U = \mathrm{LCA}(C, c(z))$ properly loses z. Below U, a node covers z exactly when z is an endpoint of its pair, a constant-time test. If $C = U$ or $z \in C$ for real C, return $\mathrm{depth}(C)$. Otherwise climb from C, skipping a block whenever its landing is strictly below U and still omits z. At the first parent equal to U or containing z, return that parent's depth. This computes the cutoff in $O(L)$ time for each remaining predecessor.

The real ancestors of C all properly lose m_y. Consequently the depth of B_* in Lemma 3.1 is the minimum of $\mathrm{depth}(C)$ and all the other predecessors' cutoffs. One ancestor lookup retrieves B_*. Each distinct predecessor is processed once at its head, so the total work is $O((n_H + m_H)L)$. The temporary rows and permanent node arrays give the stated space bounds. □

4. Transferring exits between nested cuts

The forest tells us whether deleting two *whole* blocks loses a target. A physical failure can lie strictly inside either block, in which case some of its exits remain usable. The following identity computes exactly which original tails have to be tested. We use the notation $X_s^B(y)$ of Section 2 in a fixed quotient H_d.

Theorem 4.1 (exit transfer). Let P be a real child of a real pair B in the forest of H_d. For each $s \in B$ and $y \in J(P)$,

$$X_s^B(y) = \bigcup_{p \in P \smallsetminus B} X_s^B(p) \;\cup\; \begin{cases} X_s^P(y), & s \in P, \\ \varnothing, & s \notin P. \end{cases}$$

The union over fresh endpoints is independent of y.

Proof. Lemma 3.1 puts y and every fresh endpoint $p \in P \smallsetminus B$ in $J(B)$, so all the sets on the left and in the union are defined. For each fresh p, deletion of the other endpoint of P preserves y. A surviving path must visit p, since deleting both endpoints loses y. Its suffix from p to y avoids the two earlier endpoints of B. Appending the suffix to any exit witnessing $X_s^B(p)$ proves that this tail belongs to $X_s^B(y)$.

For the inherited term, write $B = \{s, v\}$ and $P = \{s, p\}$. Take an arc out of s whose continuation to y avoids P. If that continuation visits v, choose a source path to v avoiding s, possible by flatness. The path also avoids p, which follows v. Concatenating this prefix with the continuation from v would reach y after deleting P. Thus the continuation also avoids v, and the original tail belongs to $X_s^B(y)$.

Conversely, take an arc witnessing $X_s^B(y)$. If its continuation visits a fresh endpoint p, the prefix of that continuation to p places the tail in $X_s^B(p)$. Otherwise the continuation avoids P, since it already avoids all of B. For $s \in P$ this gives the inherited term. For $s \notin P$, there exists a source path to s avoiding P. Every fresh endpoint follows s, and the at most one shared endpoint cannot by itself dominate s. Concatenation with the chosen arc and continuation would survive P, a contradiction. This also covers disjoint parent and child pairs. □

A graph that needs both terms. Take the seven vertices r, U, V, ℓ, h, p, y and exactly the arcs in these three paths:

$$r \longrightarrow U \longrightarrow \ell \longrightarrow p \longrightarrow y, \qquad r \longrightarrow V \longrightarrow p, \qquad U \longrightarrow h \longrightarrow y.$$

Figure 1 pairs the original DAG with its quotient H_r. The arcs from U in the quotient carry their original tail labels.

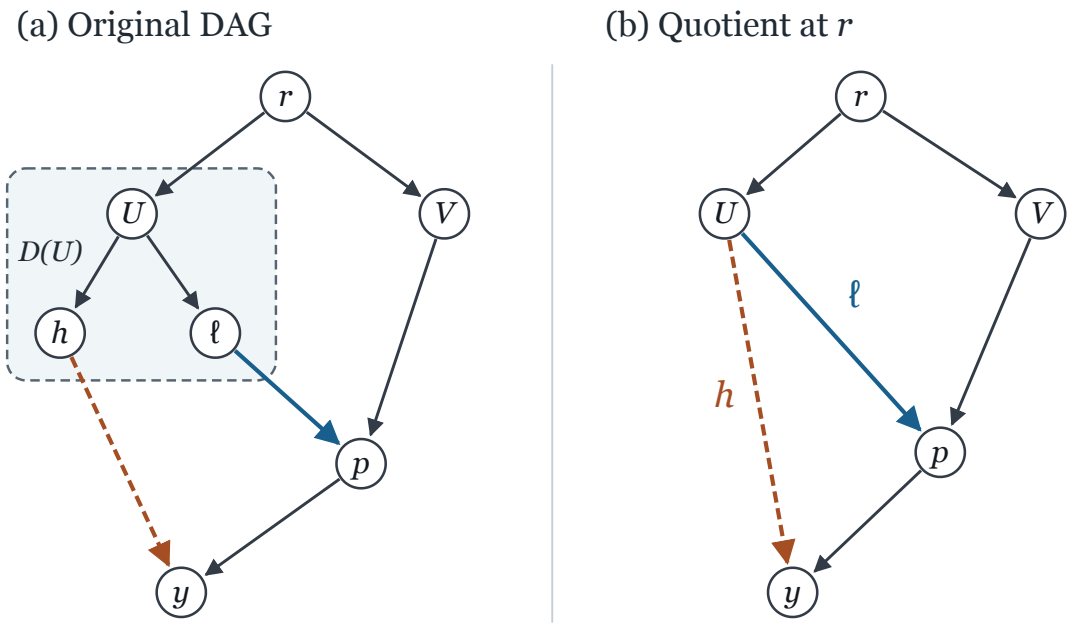

Figure 1: **Figure 1.** The original DAG and its quotient at r. The exit labelled ℓ reaches the fresh endpoint p. The exit labelled h reaches y directly and contributes through the shared endpoint U.

The block $D(U)$ contains ℓ and h. In H_r, the pair $B = \{U, V\}$ properly loses p and y. Its child $P = \{U, p\}$ properly loses y. For the shared endpoint U, the first term of the identity contributes the physical tail ℓ on the exit to p. The inherited term contributes h on the direct exit to y. Deleting h, V leaves the route through ℓ, whereas deleting ℓ, V leaves the route through h. Deleting U, V removes both. This is why storing only the separating pair, or keeping only one of the two terms, loses information about the original target.

5. Building and querying the index

Lemma 2.2 needs only the smallest and largest preorder ranks of the exit tails. For two nonempty intervals define

$$[l, h] \sqcup [l', h'] = [\min(l, l'), \max(h, h')].$$

The empty interval is the identity. Then $\sigma(X \cup Y) = \sigma(X) \sqcup \sigma(Y)$, including when either set is empty. This associative operation takes constant time on rank pairs. We can therefore apply Theorem 4.1 directly to the summaries.

Lemma 5.1 (compressed exit summaries). Given the completed forest and its jump pointers in H_d, the original-tree preorder ranks, and the labelled quotient arcs, one can build a base vector of two intervals for each vertex with a real label and two constant-size maps for each real pair node. For any $y \in J(B)$ these records recover both intervals $\sigma_s^B(y)$, $s \in B$, in $O(L)$ time. Building the records takes $O((n_H + m_H)L)$ time and $O(n_H + m_H)$ working words. They occupy $O(n_H)$ words after construction.

Proof. Order the endpoints in each pair by topological number. For a real forest edge from child P to parent B, form a map with two output coordinates. In coordinate $s \in B$, its constant is the join of $\sigma_s^B(p)$ over the fresh endpoints p. If $s \in P$, join that constant with the input coordinate for s. Otherwise return the constant. Theorem 4.1 proves the map's effect for every target below P. In particular its constants can be shared. Composing two maps of this form again gives two such coordinates. For a coordinate that copies an intermediate input, join the outer constant with the corresponding inner constant and retain the inner input slot if present. A coordinate with no input slot keeps its outer constant. Composition requires only a constant number of interval joins.

Process the vertices of H_d in topological order. Store at y the two summaries for its deepest pair $c(y)$ when that pair is real. When a pair P first appears at such a vertex y, install its map to its parent B *before* computing the base at y. If B is real, each fresh endpoint p precedes y and lies in $J(B)$. Its base at $c(p)$ is already present. Every real pair on the path from $c(p)$ up to B properly loses p, so that pair first appeared no later than p. Evaluating the maps along this path gives the constants needed for the new edge. A map to the artificial root may return two empty coordinates since no exit query asks for that root.

Now let $B = c(y)$ be real. Inspect every entering arc $z \to y$. If $z = s \in B$, join the singleton interval of its original tail's preorder rank into coordinate s. Otherwise $z \in J(B)$, since an unfailed route to z would extend to y after deletion of B. Evaluate the stored maps from $c(z)$ to B on the base at z, and join both coordinates into the base at y. Every admissible exit is included by considering the last arc of its witnessing path. Every contribution extends to a witnessing path. The predecessor z is earlier than y, so all these reads are available.

For each real node v store the map to its parent p and the map to its jump ancestor. If $j(v) = p$, the maps coincide. In the merge case of Lemma 3.2, compose, in order of traversal, the edge map from v to p, the jump map from p to $a = j(p)$ and the jump map from a to $j(a)$. The jumps start at proper ancestors of v. Those pairs were already present when the sweep of Lemma 3.1 first encountered v, so their maps are available in the same topological order. Thus installing the jump map takes constant time once the edge map is known. To evaluate a base at a specified ancestor, follow the jumps and parent steps of Lemma 3.2, applying one map at each move. This takes $O(L)$ time.

Each entering arc requires at most one such evaluation, and each new pair needs at most two to obtain its fresh-endpoint constants. Initializing the bases and storing the maps takes constant time per record. Lemma 3.1 bounds the number of pair nodes by n_H, giving the construction and space bounds. □

Lemma 5.2 (constant-time lookup on the completed forest). For each flat quotient, $O(n_H)$ additional words built in $O(n_H L)$ time suffice for the following queries in $O(1)$ worst-case time. Given distinct nonroot u, v, y, test whether $y \in J(\{u, v\})$ and, if so, return its pair node and both exit intervals. Given a real ancestor pair A of a real pair C, also return the first edge below A on which each endpoint of A disappears, if it disappears by C.

Proof. Equip the completed forest, with dummy root 0, with static lowest-common-ancestor and level-ancestor indexes [BFC00, BFC04]. Each uses linear construction time and space and returns its answer in constant time. Write $\mathrm{LA}(v, h)$ for the ancestor of v at depth h.

Finding the pair. Order u, v by topological rank with $u < v$. If $B = \{u, v\}$ properly loses any vertex, the first such vertex has predecessor set $\{u, v\}$. Its construction in Lemma 3.1 attaches B to an ancestor of $c(v)$. The node B cannot itself be an ancestor of $c(v)$, since $v \in B$ excludes $v \in J(B)$. Let $U = \mathrm{LCA}(c(v), c(y))$. If B is an ancestor of $c(y)$, its parent must be U. Thus when $c(y) \neq U$ it suffices to inspect the child

$\mathrm{LA}(c(y), \mathrm{depth}(U) + 1)$. It has endpoints u, v exactly when $y \in J(B)$. When $c(y) = U$, there is no qualifying child.

Finding a dropped endpoint. Suppose $s \in X$ disappears at a child Y. If s reappeared at a descendant of Y, nesting would put s in $Z(Y) \smallsetminus Y = J(Y) \subseteq J(X)$, contradicting $s \in X$. Thus an endpoint occurs on a contiguous portion of any descending path. For each occurrence of s in a real pair X, store its *birth depth*, the depth of the highest consecutive ancestor carrying s. This is copied from the parent occurrence when it exists and otherwise set to the current depth.

On each edge $X \to Y$ record the endpoints of X missing from Y and their birth depths. Give the edge weight equal to the least recorded depth or $+\infty$ for an empty record. Edges out of 0 also have weight $+\infty$. For an ancestor A and descendant C, inspect the edge range

$$(A, C] = \{\, X \to Y : A \prec Y \preceq C \,\},$$

where $\preceq$ is inclusive forest ancestry. This range starts with the edge **leaving** A and is empty for $A = C$. A record on the range names a dropped endpoint of A exactly when its birth depth is at most $\mathrm{depth}(A)$. At most two edges have a qualifying record. Find an edge of minimum weight in the full range. If it qualifies, report each record on that edge whose birth depth meets the threshold. Repeat the minimum test on each of the two remaining edge ranges, ignoring empty ranges. Three minima find all drops, including two at the same edge. For example, if r feeds a, b, both feed x, y, and x, y feed t, the pairs $\{a, b\}$ and $\{x, y\}$ are parent and child. Both original endpoints disappear on that single edge.

Reading the exit intervals. For each real pair X, create one node for each of its two endpoint occurrences. For a child Y and each $s \in X$, join the occurrence at X to its occurrence at Y if retained, or to a new leaf for the drop $(X \to Y, s)$ otherwise. Assign this edge the constant for s in the map of Lemma 5.1. Attach every occurrence born without a parent occurrence to a common root with an empty-interval edge. This tree has at most two occurrence nodes and two drop leaves per real pair, in addition to its root.

Now take $A = \{u, v\}$, $y \in J(A)$ and $C = c(y)$. For an endpoint $s \in A$ that persists to C, join the intervals on the path between its two occurrences and its base interval at y for that slot of C. For an endpoint that drops, join the intervals from its occurrence at A through the drop leaf found above. Theorem 4.1 gives exactly $\sigma_s^A(y)$ by this rule. At a retained occurrence the map adds its edge constant and inherits the next value. At a drop its constant is the whole value. A path with no edges contributes the empty interval.

It remains to implement the extrema. For a tree with N nodes and weighted edges, process the edges in decreasing weight, breaking ties by edge ID. When an edge joins two components, make their two current roots the children of a new node labelled by that edge. For distinct original nodes, their LCA in this reconstruction tree identifies an edge of minimum weight on their original path, since this is the edge that first joins the two components. Increasing weight gives a maximum. Sorting and disjoint-set unions take $O(N \log(N + 1))$ time and $O(N)$ words, including the linear-space LCA index [DLW14, BFC00]. Use this construction once on drop weights and twice on the occurrence tree for lower and upper interval endpoints. Empty intervals contribute neutral extrema and can be encoded by ranks outside the valid range. In each nonempty range the LCA identifies an edge as well as its value, so the two ranges left by a drop are available from its endpoints. The number of lookups per query is bounded by an absolute constant. □

Proof of Theorem 1.1

Apply the reachable-subgraph and reducibility reduction in Section 2, then topologically order the DAG. Equip $\mathcal{D}$ with preorder intervals, LCA and level ancestor. For an arc $x \to y$ let $d = \mathrm{idom}(y)$. When $x \neq d$, the child below d containing x is $\mathrm{LA}(x, \mathrm{depth}(d) + 1)$. An array gives each child its local quotient index, so one pass assigns all labelled arcs. The quotient vertex counts sum to at most $2n$, and the quotient arc counts sum to at most m. Lemmas 3.2, 5.1 and 5.2 together cost $O((n + m)L)$ time and $O(n + m)$ peak words.

Translate the query IDs. Source failure and initial target unreachability settle the answer immediately. If all failed IDs were discarded or the failure set was empty, the retained target survives. Use ordinary ancestry for one failure or two comparable failures. Otherwise find d, the LCA of the two failures, and their child blocks u, v. Lemma 2.2 decides targets outside $D(d)$, at d, or inside either failure block using only ordinary ancestry.

For a target in a third block rooted at y, use Lemma 5.2. If y survives deletion of the two block roots it is reachable. Otherwise test the two returned intervals by Lemma 2.2, matching each coordinate to its failed vertex. This costs constant time, including for a failed target. Store the inverse dominator preorder in one array for Section 6, then discard the arc lists and construction arrays. The ID translations, dominator and pair trees, node bases and static lookup structures use $O(n)$ words in total. □

Proposition 5.3 (the logarithmic-query DAG implementation). On a DAG with specified source there is a deterministic word-RAM implementation that, from the explicit adjacency lists, builds an index in $O((n+m)L^2)$ operations with $O(n+m)$ peak words, retains $O(n)$ words, and decides any later two-failure reachability query in $O(L)$ operations from the index alone.

Proof. In Lemma 3.2 one may instead binary-search the depth of B_* and rescan the distinct predecessors at each probe, testing endpoint equality or ancestry of their labels with skew jumps. This gives the stated construction bound. Store the finished forest's preorder intervals and sorted pair keys for $O(L)$ lookup. In the ordinary dominator tree mark an edge to a largest child subtree at each nonleaf. Ascending between the resulting marked paths to find an LCA crosses $O(L)$ unmarked edges because each such ascent at least doubles subtree size. Sorted child intervals locate a block by binary search. Use Lemma 5.1 to read the two exit intervals using jumps, then apply Lemma 2.2. These operations require $O(L)$ time per query and the retained arrays use linear space. This is the resource contract proved for the word-RAM implementation in [ShaL26]. □

6. Two pivots, a sparse subgraph and a third failure

Sometimes the query asks whether *any* member of a set X survives. For two failures this disjunction can be represented by at most two vertices, which we call pivots. They may lie upstream of X. Their second property, that deleting them preserves the largest possible reachable side among cuts of X of size at most two, will let us select representatives that actually belong to X.

Lemma 6.1 (two-pivot summaries). On the completed index of Theorem 1.1, a set X of bounded cardinality has a summary $C_2(X)$ of at most two vertices computable in $O(1)$ time. For every $|F| \le 2$,

$$X \cap R(F) \neq \varnothing \quad \Longleftrightarrow \quad C_2(X) \cap R(F) \neq \varnothing.$$

Moreover, $X \cap R(C_2(X)) = \varnothing$, and every F with $|F| \le 2$ that disconnects all of X satisfies $R(F) \subseteq R(C_2(X))$. Two summaries can be merged in $O(1)$ time. Thus an arbitrary list X can be summarized in one pass in $O(1+|X|)$ time and $O(1)$ additional words.

Proof. Work on the DAG from Section 2. Remove initially unreachable members of X. Return the empty set for an empty input. Otherwise let d be the LCA of X in $\mathcal{D}$. If $d \in X$, return $\{d\}$. Every path to X visits d, and every cut killing d leaves a reachable side contained in $R(\{d\})$. This includes $d = r$.

Choosing the quotient pair. Suppose $d \notin X$. Let Y be the child roots of d whose subtrees meet X. There are at least two. A cut killing X while leaving d alive must have two failures strictly below d. Paths from d to its descendants can stay inside $D(d)$, and one internal failure cutting all of X would be a deeper common dominator. They must occupy different child blocks. If both were in one block, another member of Y could be reached avoiding that whole block, and hence so could a target below it. Replacing both failures with their block roots gives a quotient pair cutting all of Y.

Find a quotient cut P whose reachable side contains that of every quotient pair cutting Y. If $|Y| = 2$, take $P = Y$. Every path surviving a cut of Y avoids Y. If $|Y| \ge 3$, the latest root v in Y must be a proper loss of every such pair. A pair containing v could lose at most one earlier root, since every proper loss follows both endpoints. Thus all candidate pairs lie on the path to $C = c(v)$. The deepest one covering all of Y has the smallest lost set, by nesting. Its depth is $\min_{z \in Y \setminus \{v\}} r_C(z)$. A zero means that no internal cut exists. In this case return $\{d\}$, since every cut under consideration must kill d.

Each cutoff now takes constant time on the *completed* forest. Let $U = \mathrm{LCA}(C, c(z))$. If $U = C$, return its depth. Otherwise let A be the child of U toward C. Below U only endpoint membership can cover z. If $z \notin A$, return the depth of U. If $z \in C$, return the depth of C. In the remaining case Lemma 5.2 returns the

edge $X' \to Y'$ on which z drops. Return the depth of X'. One level-ancestor lookup at the minimum cutoff gives P.

Lifting the pair to the original graph. Write $P = \{u, v\}$. For each $x \in X$ lying in an endpoint block, put $\mathrm{tin}(x)$ on that side. For x in a third block y, add both exit intervals for y at P to their respective sides. By Lemma 2.2, failures $a \in D(u), b \in D(v)$ kill all of X exactly when the ranks on each side lie inside its corresponding failed subtree. Both sides receive at least one rank. Either both blocks meet X, or a properly lost block supplies nonempty exit intervals on both sides. On each side take the dominator LCA of the vertices at its least and greatest rank. Its subtree is the smallest that covers all required ranks, since dominator subtrees are preorder intervals. Call the resulting pair $C_2(X)$. It kills X and preserves every vertex preserved by a separating pair projecting to P, since the selected endpoints descend from the corresponding endpoints of that pair.

We show that this choice is furthest even among cuts with other projections. Protect d and split each proper descendant into an incoming and an outgoing copy joined by a unit arc. Direct original arcs between the corresponding copies and draw an arc to a new sink from each outgoing copy of X. Give all these connectors capacity three. Work inside $D(d)$ with source d. The minimum cut has capacity two because P separates and no one internal vertex does. There is a minimum cut with largest reachable source side [BCR18, §4.2]. Indeed, the union of two reachable minimum-cut sides is reachable inside the union. Submodularity of outgoing cut capacity and the lower bound of two for their intersection show the union also has capacity two. Taking all such unions yields a cut consisting of two split arcs. Its original vertices form a pair F_* whose reachable side contains that of every internal cut of size two. Outside $D(d)$ every vertex has a source path avoiding d, so the same inclusion holds globally.

In particular, every vertex surviving P survives F_*. Hence the endpoints of F_* are dead under P. Their projected roots are also lost in the quotient under P, since a surviving root outside the two failed blocks retains its entire block. Let P_* be those two projected roots. It cuts Y. Every quotient path surviving P avoids P_*, and the choice of P gives the reverse inclusion of reachable sides. These inclusion-minimal cuts with the same reachable side are equal. Indeed, each cut vertex has an incoming arc from the reachable side by minimality, and every arc leaving that side ends in the cut. Thus F_* projects to P. Our construction within those two blocks preserves its entire surviving side. Finally, if F kills d, its surviving paths avoid $D(d)$ and hence survive the selected internal pair. This proves the required inclusion for every admissible F.

Every path to X meets $C_2(X)$. Conversely, if F kills X, the inclusion just proved implies that F kills every pivot. This proves the Boolean equivalence. Given summaries for two sets, take the summary of the at most four pivots in their union. The equivalence gives exactly the same failure sets killing this union as kill the original union, so the largest-side property also carries over. A merge uses a bounded number of static lookups, including one inverse-preorder lookup for each extreme rank. Repeatedly merging a singleton into the current summary gives the one-pass bound. □

Corollary 6.2 (a four-incoming-arc preserver). In $O((n+m)L)$ deterministic time and $O(n+m)$ working words one can select at most $4(n-1)$ original arcs preserving $R(F)$ for every $|F| \le 2$. Each vertex has at most four incoming arcs in the subgraph. Some DAGs require four at a single vertex.

Proof. On the reduced DAG, stream the initially reachable incoming tails of each head. Keep a subset B representing their reachability disjunction. After adding a distinct tail, test each $x \in B$ by deleting the pivots of $B \setminus \{x\}$. If x survives, these failures isolate it from all other members. If some set of at most two failures isolates x, the largest-side property in Lemma 6.1 guarantees that x survives this test. Remove x when it fails the test. This preserves the disjunction, and an isolation witness remains valid when other elements are removed.

The improved anti-isolation lemma [HSW25, Lemma 2.3] bounds an irredundant list by $2^2 = 4$. To apply its edge-failure statement, split each nonroot vertex, translate an isolating vertex failure into deletion of its split arc, and observe each listed tail at its outgoing copy. An isolating witness leaves the source alive. A row with the source as a tail may keep the direct source arc alone, since every surviving path starts at the source. Thus an insertion tests at most five candidates, using constant time per test. Retain one actual input arc for each selected tail, choosing any one parallel copy. For an unfailed head other than the source, reachability is

the disjunction over its incoming tails. A failed head is unreachable in both graphs. Topological induction shows that the retained arcs give the original reachability at every vertex under each failure set. The input is scanned once after building the index.

To force four arcs, let r have two children with two leaves each, and draw an arc from each of the four leaves to a common head. Fail a leaf's sibling and the opposite branch root. Its arc is then the only surviving way into the head. Each of the four arcs is forced by one such choice. Baswana, Choudhary and Roditty give the indegree bound 2^k for general digraphs in $O(2^k mn)$ construction time [BCR18, Theorem 2]. The construction here obtains the bound for $k = 2$ within the preprocessing time of Theorem 1.1 on its graph classes. □

Proof of Corollary 1.3

Handle $r = t$, initial unreachability and failure of either endpoint directly. Two searches find the vertices lying on an r-to-t path. Topologically rank these vertices and ignore failures outside them. Write $a < b$ for increasing rank, and let $h_t(x)$ be the largest rank of a head of an outgoing arc from x in this subgraph, defaulting to the rank of x. All these maxima take one arc scan. For three distinct remaining internal failures $a < b < c$,

$$t \in R(\{a, b, c\}) \quad \Longleftrightarrow \quad \exists x \in R(\{a, b\}) \text{ on an } r\text{-to-}t \text{ path with } \operatorname{rank}(x) < \operatorname{rank}(c) < h_t(x).$$

A surviving path has consecutive vertices strictly on opposite sides of c. Conversely, take a surviving prefix to such an x, the arc to a head attaining $h_t(x)$, and a path from that head to t. The prefix precedes c, and the suffix follows all three failures. The strict inequalities exclude arcs landing on or leaving the failed vertex.

Build the two-failure index on the DAG. In a balanced segment tree on the ranks, cover the open rank interval $(\operatorname{rank}(x), h_t(x))$ by its maximal contained tree intervals. There are $O(L)$ per x, with at most two per level. Merge the singleton x into the summary at each such node. Once all intervals are installed, propagate each parent's summary into its two children, proceeding from root to leaves. By Lemma 6.1, the summary at rank j represents exactly the disjunction over tails whose intervals contain j, for every pair of failures. The chosen pivots may lie upstream of these tails.

The $O(nL)$ insertions and $O(n)$ propagations each cost constant time. Retain at most two pivot IDs per rank, the rank map and the two-failure index. With three relevant failures, probe the pivots at the rank of c under $\{a, b\}$ and take the disjunction. With fewer, query t itself in the pair index. The temporary segment tree uses $O(n)$ words. Two fixed targets need two sets of rank summaries and may share the original two-failure index. □

For fixed source and sink in arbitrary digraphs, Ahi et al. give an $O(n \log n)$-space oracle with $O(\log n)$ query time for any constant number of edge failures [ACPPS26, Corollary 1.6]. Splitting internal vertices transfers that guarantee to vertex failures. Corollary 1.3 gives linear space and constant query time for three vertex failures on DAGs and reducible flowgraphs. Its interval endpoints $h_t(x)$ depend on the target chosen in advance.

7. Guarded decoders and higher sensitivity

A guarded Boolean decoder supplies one application in which a list of targets is more natural than a single target. Its DAG has a root, sinks of constant value zero, labelled terminal sinks, OR gates, and guards. Any number of terminals may carry the same label i, denoted h_i. A guard has one child and a set of labels. For an input set K, a terminal h_i is true if $i \in K$, and a guard passes its child's value if its label set is disjoint from K. Edges point from a gate to its children. Let G_{dec} count the declared labels, vertices, arcs, guard memberships and output root.

The reference value is the output at $\{t_0\}$, for a specified label t_0. In the *healthy-zero* case this value is zero. A detecting set changes the output from its reference value, so here it makes the output one. Admissible test sets are nonempty and must contain a prescribed Q, either $\varnothing$ or $\{t_0\}$. Delete once all guards meeting Q, with their incident arcs. The *residual multiplicity* μ is the maximum, over labels $i \neq t_0$, of the number of remaining guards containing i, with maximum zero for an empty candidate set.

Corollary 7.1 (a guarded output of residual multiplicity two). For one supplied healthy-zero guarded output with $\mu \le 2$, the existence of an admissible detecting set is decidable in $O(G_{\mathrm{dec}} \log(G_{\mathrm{dec}} + 2))$ time and $O(G_{\mathrm{dec}})$ working words. A positive answer identifies an admissible detecting set.

Proof. A true output has a path to a true terminal on which every guard passes. To obtain the path choose a true child at each OR and follow the unique child of each true guard. Conversely any such path suffices. If K makes the output true by reaching h_i, the same path is true on the subset $Q \cup \{i\}$. The label i differs from t_0, since a witnessing path ending at h_{t_0} would already be true on the reference singleton.

If the common deletion removes the root, detection is impossible. Otherwise use the surviving root as source in the residual DAG. For each $i \neq t_0$, take the remaining guards containing i as the failed vertices, and take all terminals of label i as its target list. A positive list gives the test $Q \cup \{i\}$. Each remaining guard membership and terminal is scanned once in building the lists, and the number of queries is at most the number of declared labels. Both $n + m$ and $q + M$ of the resulting batch are $O(G_{\mathrm{dec}})$. Corollary 1.2 therefore gives the claimed time and space. □

For the drive-one first-stage bridge in a priority controller, the reachable-set theorem of [Sha26, Theorem 4.1] identifies $Q = \{t_0\}$ for one request bus and $Q = \varnothing$ for at least two. It also realizes an admissible detecting set by primary inputs. Corollary 7.1 concerns the decoder-level computation for a supplied healthy-zero output. Recovering circuit roles from an unannotated formula is a separate problem.

A region with two entries

Let H be a set of vertices excluding r, and suppose every arc into H from outside ends at one of two specified entries $E \subseteq H$ (one entry is also allowed). Fix these entries before preprocessing. For any $|F| \le 3$, Corollary 1.3 gives both entry statuses in constant time. If $|F \cap H| \le 2$, then for $t \in H$

$$t \in R(F) \quad \Longleftrightarrow \quad \text{some } e \in E \cap R(F) \text{ reaches } t \text{ in } G[H] - (F \cap H).$$

Indeed, use the last entry on a surviving path. In the opposite direction concatenate a surviving prefix to e and an internal suffix to t. Build a two-failure index on the induced DAG from each entry. At most two probes per listed target suffice after the entry statuses are known. For a common region the total time over q lists and M target occurrences is $O((n + m)L + q + M)$, including all preprocessing, reading the triples and combining the answers.

One case guaranteeing the internal-fault condition is a set F of up to three nonroot vertices reached by paths disjoint outside r, one path to each failed vertex. Paths to faults inside H use distinct first entries. Each such entry either is the fault itself or survives, since the other failed vertices lie on their own disjoint paths. Consequently $F_H = (F \cap H) \cup (E \setminus R(F))$ also satisfies $|F_H| \le |E|$ and $R(F_H) \cap H = R(F) \cap H$. A path surviving F avoids F_H. Conversely a path into H surviving F_H has a last entry that survives F, followed by an internal suffix avoiding F.

Handle the source and initially unreachable targets directly. If the entries are unspecified, seek a common separator for the union of the remaining targets. Split non-source vertices with unit-capacity arcs, give original connectors capacity three, and join the target out-copies to a private sink by capacity-three arcs. At most three unit augmentations either certify that there is no separator of size at most two or identify one from the final residual reachable side. For an empty target union the answers are immediate. Otherwise let E be the separator and take H to be the initially reachable vertices lost on deleting E. Its entering arcs all end at E. The network construction, searches, region recovery and list handling add $O(n + m + q + M)$ work. For unlinked faults the condition $|F \cap H| \le 2$ is tested on this recovered region. On reducible inputs use the DAG from Section 2 throughout.

Higher sensitivity

Question 7.2 (higher sensitivity). For arbitrary $\mu \ge 3$, can a deterministic combinatorial algorithm decide whether any listed target survives in any query on a DAG in $\widetilde{O}(2^{\mu} S_{\mu})$ total time, where $S_{\mu} = n + m + q + \sum_i (|F_i| + |T_i|)$ and $|F_i| \le \mu$?

For zero failures a single search suffices. For one failure a dominator tree and its subtree intervals give a linear total bound with a linear-time dominator construction [BGK08]. Corollary 1.2 handles two failures at arbitrary depth and returns every list answer. At three failures, different targets can induce different families of intervals in Corollary 1.3. The cost of constructing or sharing their summaries for arbitrary target lists remains open. A k-failure reachability preserver on a general digraph has $O(2^k n)$ arcs [BCR18], but traversing it separately may incur this cost for each distinct failure set.

There is also a limit to using the same pair forest on irreducible inputs. In the graph with arcs $r \to a, r \to b, a \to x, b \to y, x \to y, y \to x$, the dominator tree is flat. Deleting a, b loses both x and y, although neither has exactly a, b as its predecessors. Thus the first-proper-loss step of Lemma 3.1 requires the acyclic quotient.

Computational checks and formalization

On all 191,294 topologically ordered simple DAGs through seven vertices with a flat dominator tree and source first, explicit enumeration produced 42,232 instances of the exit-set identity in Theorem 4.1. All agreed as sets of exits. Both terms were indispensable in 20,052 coordinates. The logarithmic-query DAG construction agreed with direct deletion search on 421,402 choices of graph, source, target and failure set on all 1,099 ordered simple DAGs through five vertices.

For the constant-time lookup, 593,616 query answers agreed with direct search on exhaustive and sampled DAGs. The fixed-target construction agreed on 172,151 reachability tests with up to three failures. For the two-pivot summary, 786,592 disjunction tests and 316,047 comparisons of reachable-side inclusion agreed. An end-to-end test enumerated all 16,585 choices of a labelled loopless digraph through four vertices and a source, keeping the reducible inputs and adding two larger cyclic multigraph examples. On these inputs the two-failure oracle agreed with search on the original graph in 530,773 queries, the fixed-target oracle in 723,166 queries with up to three failures, and the preserver in 132,993 comparisons of whole reachable sets.

These Python checks exercise correctness. The programs use explicit dominator sets and tree lookup tables that can occupy superlinear space. The space and worst-case time bounds in Theorem 1.1 and Corollary 1.3 follow from the constructions and linear-space static tree primitives.

The published Lean 4 package [ShaL26] proves the DAG implementation in Proposition 5.3, including correctness, construction time, peak and retained words, and query time. Its selected declaration is `PaperB.BytecodeContract.twoFailureOracleBytecode`, with the standard axioms of propositional extensionality, classical choice and quotient soundness. The Lean 4 formalization of the main oracle theorem is described in the software declaration below.

Declarations

Software availability. Theorem 1.1 is formalized in Lean 4 as `PaperB.Palomar.twoFailureOracleV14`. The registered declaration covers the correctness and word-RAM bounds of the single-source two-failure oracle on DAGs and reducible flowgraphs. The Apache-2.0 source is available at the registered Git revision. The formalization is recorded in Palomar, version 1.

Funding. No external funding was received for this research.

Competing interests. The author declares no competing interests.

AI assistance. OpenAI Codex and ChatGPT assisted with mathematical exploration and manuscript preparation. The author reviewed the AI-assisted material and takes full responsibility for the manuscript.

References

- [ACPPS26] M. Ahi, K. Choudhary, S. Pande, Pushpraj and L. Saggi. Maximum-flow and minimum-cut sensitivity oracles for directed graphs. *ITCS 2026*, article 5. Article
- [BCR18] S. Baswana, K. Choudhary and L. Roditty. Fault-tolerant subgraph for single-source reachability: general and optimal. *SIAM Journal on Computing* 47, 2018. Article
- [BFC00] M. A. Bender and M. Farach-Colton. The LCA problem revisited. *LATIN 2000*, 88–94. Article

- [BFC04] M. A. Bender and M. Farach-Colton. The level ancestor problem simplified. *Theoretical Computer Science* 321, 5–12, 2004. Article
- [BGK08] A. L. Buchsbaum, L. Georgiadis, H. Kaplan, A. Rogers, R. E. Tarjan and J. R. Westbrook. Linear-time algorithms for dominators and other path-evaluation problems. *SIAM Journal on Computing* 38, 1533–1573, 2008. Article
- [Cho16] K. Choudhary. An optimal dual fault tolerant reachability oracle. *ICALP 2016*, article 130. Article
- [Cho17] K. Choudhary. *Compact and Efficient Fault Tolerant Structures for Directed Graphs*. PhD thesis, IIT Kanpur, 2017. Theorem 3.2
- [DLW14] E. D. Demaine, G. M. Landau and O. Weimann. On Cartesian trees and range minimum queries. *Algorithmica* 68, 610–625, 2014. Article
- [GT15] L. Georgiadis and R. E. Tarjan. Dominator tree certification and divergent spanning trees. *ACM Transactions on Algorithms* 12, article 11, 2015. Article · Author version, §3
- [HSW25] G. Hoppenworth, T. Saranurak and B. Wang. Near-optimal fault-tolerant strong connectivity preservers. *FOCS 2025*. Article · Lemma 2.3, author version
- [HU74] M. S. Hecht and J. D. Ullman. Characterizations of reducible flow graphs. *Journal of the ACM* 21, 367–375, 1974. Article
- [Kar88] K. Karplus. *Representing Boolean Functions with If-Then-Else DAGs*. Technical report UCSC-CRL-88-28, University of California, Santa Cruz. Report, §§2.3–2.4
- [LT79] T. Lengauer and R. E. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Transactions on Programming Languages and Systems* 1, 121–141, 1979. Article
- [Mye83] E. W. Myers. An applicative random-access stack. *Information Processing Letters* 17, 241–248, 1983. Article, §§3–4
- [Sha26] I. A. Shaik. *Polynomial Fault Detection Without Bounded Width: Exact Reachable Sets in Priority Controllers*. Version 1.1, 2026. Preprint
- [ShaL26] I. A. Shaik. *Reachability After Two Vertex Failures in DAGs*. Lean 4 formalization of version 1.3, Theorem 1.1, 2026. Pinned theorem and verification record · Version 1.3 of the paper
- [TD15] M. Teslenko and E. Dubrova. *A Linear-Time Algorithm for Finding All Double-Vertex Dominators of a Given Vertex*. 2015. Author paper
- [TDT06] M. Teslenko, E. Dubrova and H. Tenhunen. An efficient algorithm for computing common double-vertex dominators. *IADIS International Conference Applied Computing*, 34–41, 2006. Publisher record
- [Tes08] M. Teslenko. *All Around Logic Synthesis*. Doctoral thesis, KTH, 2008. Chapter 3