Testing 2-vertex connectivity and computing pairs of vertex-disjoint *s*-*t* paths in digraphs*

Loukas Georgiadis

Department of Informatics and Telecommunications Engineering, University of Western Macedonia, Greece. E-mail: lgeorg@uowm.gr

Abstract. We present an $O(m+n)$-time algorithm that tests if a given directed graph is 2-vertex connected, where m is the number of arcs and n is the number of vertices. Based on this result we design an $O(n)$-space data structure that can compute in $O(\log^2 n)$ time two internally vertex-disjoint paths from s to t, for any pair of query vertices s and t. The two paths can be reported in additional $O(k)$ time, where k is their total length.

1 Introduction

A directed (undirected) graph is *k-vertex connected* if it has at least $k+1$ vertices and the removal of any set of at most $k-1$ vertices leaves the graph strongly connected (connected). The *vertex connectivity* $\kappa \equiv \kappa(G)$ of a graph G is the maximum k such that G is k-vertex connected. Graph connectivity is one of the most fundamental concepts in graph theory with numerous practical applications [3]. Currently, the fastest known algorithm for computing κ is due to Gabow [12], with $O((n+\min\{\kappa^{5/2}, \kappa n^{3/4}\})m)$ running time. In [12], Gabow also showed how to test $\alpha\delta$-vertex connectivity in $O((\kappa+\sqrt{n})\sqrt{n}m)$ time, where δ is the minimum degree of the given graph and α is an arbitrary fixed constant less than one. Henzinger et al. [18] showed how to test k-vertex connectivity in time $O(\min\{k^3+n, kn\}m)$. They also gave a randomized algorithm for computing κ with error probability $1/2$ in time $O(nm)$. For an undirected graph, a result of Nagamochi and Ibaraki [21] allows m to be replaced by κn or kn in the above bounds. Cheriyan and Reif [9] showed how to test k-vertex connectivity in a directed graph with a Monte Carlo algorithm with running time $O((M(n)+nM(k))\log n)$ and error probability $< 1/n$, and with a Las Vegas algorithm with expected running time $O((M(n)+nM(k))k)$. In these bounds, $M(n)$ is the time to multiply two $n \times n$ matrices, which is $O(n^{2.376})$ [10].

Note that for constant κ or k the above bounds are $O(nm)$ for deterministic algorithms and $O(M(n))$ for randomized algorithms. To the best of our knowledge, these are also the best previously known bounds for testing $k = 2$ for a directed graph. In Section 2 we present a linear-time algorithm for this problem. In the undirected case, linear-time algorithms were given by Tarjan [23] for

* This research project has been funded by the John S. Latsis Public Benefit Foundation. The sole responsibility for the content of this paper lies with its author.

testing $k = 2$, and by Hopcroft and Tarjan [19] for testing $k = 3$. Our algorithm is based on a new characterization of 2-vertex connected directed graphs, based on the concept of *dominators in flowgraphs*. A flowgraph $G(s) = (V, A, s)$ is a directed graph with a distinguished source vertex $s \in V$ such that every vertex is reachable from s. The *dominance relation* in $G(s)$ is defined as follows: A vertex w *dominates* a vertex v if every path from s to v includes w. We denote by $dom(v)$ the set of vertices that dominate v. Obviously, $dom(s) = \{s\}$ and $dom(v) \supseteq \{s, v\}$, for any $v \neq s$; s and v are the *trivial dominators* of v. The dominance relation is transitive and its transitive reduction is the *dominator tree* D, which is rooted at s and satisfies the following property: For any two vertices v and w, w dominates v if and only if w is an ancestor of v in D [1]. For any vertex $v \neq s$, the *immediate dominator* of v is the parent of v in D. It is the unique vertex that dominates v and is dominated by all vertices in $dom(v) \setminus \{v\}$. The computation of dominators appears in several application areas, such as program optimization and code generation, constraint programming, circuit testing, and theoretical biology [17]. There is an $O(m\alpha(m, n))$-time algorithm [20] to compute dominators that has been used in many of these applications, where $\alpha(m, n)$ is a functional inverse of Ackermann's function, which is very slow-growing. This algorithm also works very well in practice [17], even though it has some conceptual complexities. There are even more complicated truly linear-time algorithms that run on random-access machines [2, 6] and on pointer machines [15, 14, 5]. Our 2-vertex connectivity algorithm needs to test whether certain flowgraphs, derived from the input directed graph, have trivial dominators only (i.e., the immediate dominator of all vertices is the source vertex of the flowgraph). This can be done by computing the dominators of the flowgraph, but for our purpose a simpler alternative is to use a dominator-verification algorithm [16]. The algorithm given in [16] requires a linear-time solution to a special case of the disjoint set union problem [13] in order to achieve linear running time. With a standard disjoint set union structure the algorithm runs in $O(m\alpha(m, n))$ [24], and avoids the complexities of the Lengauer-Tarjan algorithm for computing dominators.

The second part of the paper (Section 3) deals with the task of computing two internally vertex-disjoint s-t paths (i.e., paths directed from s to t), for any given source vertex s and target vertex t. This problem can be reduced to computing two edge-disjoint paths (by applying a standard vertex splitting procedure), which in turn can be carried out in $O(m)$ time by computing two flow-augmenting paths [3]. In Section 3 we present a faster algorithm for 2-vertex connected directed graphs. First we note that our linear-time algorithm for testing 2-vertex connectivity allows us to find a 2-vertex connected spanning subgraph of the input directed graph with $O(n)$ arcs. Hence, the flow-augmenting algorithm can compute two internally vertex-disjoint s-t paths in $O(n)$ time. We can improve this further with the use of *independent branchings*. A branching of a directed graph G is a rooted spanning tree of G such that all vertices other than the root have in-degree one, whereas the root has in-degree zero. Two branchings of G are independent if for each vertex v, the two root-to-v paths are internally vertex-disjoint. In [16], a linear-time algorithm that constructs two

independent branchings rooted at the same vertex is presented. Based on this result, we construct a data structure that can compute two internally vertex-disjoint s-t paths, for any s and t, in $O(\log^2 n)$ time, so that the two paths can be reported in constant time per vertex.

2 Testing 2-vertex connectivity

Let $G = (V, A)$ be the input directed graph (digraph). For any vertex $s \in V$, we denote by $G(s) = (V, A, s)$ the corresponding flowgraph with source vertex s. We can assume that G is strongly connected, which implies that all vertices are reachable from s and reach s. We also let $G^r(s)$ be the flowgraph derived from $G(s)$ after reversing all arc directions. For any $u, v \in V$, the *local connectivity* $\kappa(u, v)$ of G is defined as the maximum number of internally vertex-disjoint paths from u to v. By Menger's theorem (see, e.g., [3]) this is equal to the minimum number of cut vertices in an u-v separator if $(u, v) \notin A$. The next lemma relates local and global connectivity.

Lemma 1. *(See, e.g., [3])* $\kappa(G) = \min_{u,v \in V} \kappa(u, v)$.

Thus, a 2-vertex connected digraph $G = (V, A)$ satisfies the following property.

Lemma 2. *Let $G = (V, A)$ be a 2-vertex connected digraph. Then, for any vertex $s \in V$, both flowgraphs $G(s)$ and $G^r(s)$ have trivial dominators only.*

Proof. Lemma 1 and the fact that $\kappa(G) \geq 2$ imply that, for any vertex $v \in V - s$, there are at least two internally vertex-disjoint paths from s to v. Hence s is the immediate dominator of v in $G(s)$. Similarly, there are at least two internally vertex-disjoint paths from v to s. Hence s is also the immediate dominator of v in $G^r(s)$. □

Our goal is to prove that the following property characterizes 2-vertex connected digraphs.

Property 1. Let a and b be two distinct vertices of a digraph G. Then, all four flowgraphs $G(a)$, $G(b)$, $G^r(a)$ and $G^r(b)$ have trivial dominators only.

2.1 Dominators and 2-vertex connectivity

Lemma 2 implies that Property 1 is necessary for a digraph to be 2-vertex connected. Therefore, it remains to show that Property 1 is also sufficient. First we introduce some additional notation. We denote by $P[a, b]$ a simple path (i.e., a path with no repeated vertices) from a to b, and let $P(a, b]$ denote this path excluding a. Similarly, $P[a, b)$ excludes b, and $P(a, b)$ excludes both a and b. We define the *rank* of a vertex $c \in P[a, b]$ as the number of vertices in $P[a, c]$. (There is no ambiguity in this definition since $P[a, b]$ is a simple path.) Let $P[x, y]$ and $Q[y, z]$ be two simple paths. We denote by $P[x, y] \cdot Q[y, z]$ the catenation

of these two paths (which is not necessarily a simple path). Furthermore, let $R = P[x, y] \cdot Q[y, z]$. We denote by $R[x, z]$ a simple path from x to z that can be formed from R. One way to accomplish this is as follows. We find the vertex $w \in P[x, y] \cap Q[y, z]$ with the highest rank in $Q[y, z]$. Then we let $R[x, z] = P[x, w] \cdot Q[w, z]$.

For our construction we will need a series of technical lemmas.

Lemma 3. *([16]) Consider the flowgraph $G(s) = (V, A, s)$ and a vertex $v \neq s$, such that s is the immediate dominator of v in $G(s)$. If $(s, v) \notin A$ then there are two internally vertex-disjoint paths from s to v.*

Lemma 4. *Let $G = (V, A)$ be a digraph such that for each $(u, v) \notin A$ there are two internally vertex-disjoint paths from u to v. Then G is 2-vertex connected.*

Proof. By Lemma 1 we need to show that there is a pair of vertex-disjoint paths between any pair $u, v \in V$, so we consider the case $(u, v) \in A$. Observe that for any $w \in V \setminus \{u, v\}$ there is a path $P[u, w]$ that does not contain v. This is clear when $(u, w) \in A$. If $(u, w) \notin A$ then there are two vertex-disjoint paths from u to w, so they cannot both contain v. Similarly, we have a path $Q[w, v]$ that does not contain u. Therefore (u, v) and $P[u, w] \cdot Q[w, v]$ is a pair of internally vertex-disjoint paths from u to v. □

Lemma 5. *Consider three distinct vertices a,c and d such that the following two pairs of internally vertex-disjoint paths exist: $P_1[a, c]$ and $P_2[a, c]$, and $P_3[d, a]$ and $P_4[d, a]$. If $P_3[d, a) \cap P_1(a, c] \neq \emptyset$ then there are two internally vertex-disjoint paths from d to c.*

Proof. Let x be the vertex with the lowest rank in $P_3[d, a]$ such that $x \in P_3[d, a) \cap \big(P_1(a, c] \cup P_2(a, c]\big)$. (The premises of the lemma imply that x exists.) Furthermore, we can assume that $x \in P_3[d, a) \cap P_1(a, c]$, since we can alternate the role of $P_1[a, c]$ and $P_2[a, c]$ if $x \notin P_1(a, c]$. This also implies that $d \notin P_2[a, c]$. We distinguish the following cases:

a) $P_4(d, a) \cap P_1(a, c) = \emptyset$ and $P_4(d, a) \cap P_2(a, c) = \emptyset$. (See case (a) of Figure 1 in Appendix A.) Let $R = P_4[d, a] \cdot P_2[a, c]$. Then the paths $P_3[d, x] \cdot P_1[x, c]$ and $R[d, c]$ are internally vertex-disjoint.[1]

b) $P_3(d, a) \cap P_2(a, c) = \emptyset$ and $P_4(d, a) \cap P_2(a, c) = \emptyset$. (See case (b) of Figure 1 in Appendix A.) Suppose $P_4(d, a) \cap P_1(a, c) \neq \emptyset$, otherwise we have case (a). Let e be the vertex with the highest rank in $P_1[a, c]$ such that $e \in P_3[d, a) \cap P_1(a, c]$. Also let f be the vertex with the highest rank in $P_1[a, c]$ such that $f \in P_4(d, a) \cap P_1(a, c]$. Since $P_3[d, a)$ and $P_4(d, a)$ are vertex-disjoint we have $e \neq f$. If e has higher rank in $P_1[a, c]$ than f then the paths $P_3[d, e] \cdot P_1[e, c]$ and $P_4[d, a] \cdot P_2[a, c]$ are internally vertex-disjoint. Otherwise, the paths $P_4[d, f] \cdot P_1[f, c]$ and $P_3[d, a] \cdot P_2[a, c]$ are internally vertex-disjoint.

c) $P_3(d, a) \cap P_2(a, c) \neq \emptyset$ and $P_4(d, a) \cap P_2(a, c) = \emptyset$. If $P_4(d, a) \cap P_1(a, c) = \emptyset$ then we have case (a), so suppose $P_4(d, a) \cap P_1(a, c) \neq \emptyset$. Let g be the vertex

[1] Note that we can have $c \in P_4(d, a)$, in which case R is not a simple path.

with the lowest rank in $P_3[d,a]$ such that $g \in P_3(d,a) \cap P_2(a,c]$. Also, let e be the vertex with the highest rank in $P_1[a,c]$ such that $e \in P_3[d,g] \cap P_1(a,c]$, and let f be the vertex with the highest rank in $P_1[a,c]$ such that $f \in P_4(d,a) \cap P_1(a,c]$. Suppose the rank of e in $P_1[a,c]$ is lower than that of f. (See case (c1) of Figure 1 in Appendix A.) Then the paths $P_3[d,g] \cdot P_2[g,c]$ and $P_4[d,f] \cdot P_1[f,c]$ are internally vertex-disjoint. If, on the other hand, the rank of e in $P_1[a,c]$ is higher than that of f, then the paths $P_3[d,e] \cdot P_1[e,c]$ and $P_4[d,a] \cdot P_2[a,c]$ are internally vertex-disjoint. (See case (c2) of Figure 1 in Appendix A.)

d) $P_3(d,a) \cap P_2(a,c) \neq \emptyset$ and $P_4(d,a) \cap P_2(a,c) \neq \emptyset$. Let y be the vertex with the lowest rank in $P_4[d,a]$ such that $y \in P_4[d,a) \cap \big(P_1(a,c] \cup P_2(a,c]\big)$. First suppose that $y \in P_2(a,c]$. Then the paths $P_3[d,x] \cdot P_1[x,c]$ and $P_4[d,y] \cdot P_2[y,c]$ are internally vertex-disjoint.

Now consider that $y \in P_1(a,c)$. Let g be the vertex with the lowest rank in $P_3[d,a]$ such that $g \in P_3(d,a) \cap P_2(a,c]$, and let h be the vertex with the lowest rank in $P_4[d,a]$ such that $h \in P_4(d,a) \cap P_2(a,c]$. Also, let e be the vertex with the highest rank in $P_1[a,c]$ such that $e \in P_3[d,g] \cap P_1(a,c]$, and let f be the vertex with the highest rank in $P_1[a,c]$ such that $f \in P_4(d,h] \cap P_1(a,c]$. Since $P_3[d,a)$ and $P_4(d,a)$ are vertex-disjoint we have $e \neq f$. If the rank of e in $P_1[a,c]$ is lower than that of f then the paths $P_3[d,g] \cdot P_2[g,c]$ and $P_4[d,f] \cdot P_1[f,c]$ are internally vertex-disjoint. (See case (d1) of Figure 1 in Appendix A.) Otherwise, the paths $P_3[d,e] \cdot P_1[e,c]$ and $P_4[d,h] \cdot P_2[h,c]$ are internally vertex-disjoint. (See case (d2) of Figure 1 in Appendix A.) □

Lemma 6. *Consider three distinct vertices a,b and c such that the following two pairs of internally vertex-disjoint paths exist: $P_1[a,c]$ and $P_2[a,c]$, and $P_3[b,c]$ and $P_4[b,c]$. Then there are two internally vertex-disjoint paths $Q[a,c]$ and $Q'[b,c]$.*

Proof. If $P_1[a,c) \cap P_3[b,c) = \emptyset$ then, clearly, the lemma holds with $Q[a,c] = P_1[a,c]$ and $Q'[b,c] = P_3[b,c]$. Now suppose that $P_3[b,c)$ intersects $P_1[a,c)$. Let e be the vertex with the lowest rank in $P_3[b,c]$ such that $e \in P_3[b,c) \cap \big(P_1[a,c) \cup P_2[a,c)\big)$. We can assume, with no loss of generality, that $e \in P_1[a,c)$. If $e = a$ then the paths $Q[a,c] = P_3[a,c]$ and $Q'[b,c] = P_4[b,c]$ are internally vertex-disjoint. Otherwise, if $e \neq a$, the paths $Q[a,c] = P_2[a,c]$ and $Q'[b,c] = P_3[b,e] \cdot P_1[e,c]$ are internally vertex-disjoint. □

Lemma 7. *Let a, b be any two distinct vertices of a digraph $G = (V,A)$ that satisfy Property 1. Then for any vertex $c \notin \{a,b\}$ there are two internally vertex-disjoint paths $Q[a,c]$ and $Q'[b,c]$.*

Proof. Property 1 and Lemma 3 imply that for any $c \neq a$, if $(a,c) \notin A$ then there are two internally vertex-disjoint paths from a and c. Similarly, for any $c \neq b$, if $(b,c) \notin A$ then there are two internally vertex-disjoint paths from b and c. It is clear that the lemma holds when G contains both arcs (a,c) and (b,c). Next consider that G contains the arc (a,c) but not (b,c). Then there are two internally vertex-disjoint paths from b and c, therefore they cannot both contain

a and the lemma follows. The case $(a, c) \notin A$ and $(b, c) \in A$ is symmetric. Finally, assume that $(a, c) \notin A$ and $(b, c) \notin A$. Now we have two internally vertex-disjoint paths from a to c and two internally vertex-disjoint paths from b to c, hence the result follows from Lemma 6. □

Symmetrically we also get the following results for paths entering a and b.

Lemma 8. *Consider three distinct vertices a,b and d such that the following two pairs of internally vertex-disjoint paths exist:* $P_1[d, a]$ *and* $P_2[d, a]$*, and* $P_3[d, b]$ *and* $P_4[d, b]$*. Then there are two internally vertex-disjoint paths* $Q[d, a]$ *and* $Q'[d, b]$*.*

Lemma 9. *Let a, b be any two distinct vertices of a digraph* $G = (V, A)$ *that satisfy Property 1. Then for any vertex* $d \notin \{a, b\}$ *there are two internally vertex-disjoint paths* $Q[d, a]$ *and* $Q'[d, b]$*.*

Before proceeding to our main lemma we make the following observation. Our goal is to show that for any pair of vertices d and c there are two internally vertex-disjoint paths from d to c. Unfortunately we cannot obtain the desired result immediately by applying Lemmas 7 and 9; Lemma 7 gives us two paths, from a and b to c, that only meet at c. Lemma 9 gives us two paths, from d to a and b, that only meet at d. These paths, however, do not suffice to construct two internally vertex-disjoint paths from d to c. (A counterexample is illustrated in Figure 2(a) in Appendix A.) Therefore our arguments need to be more subtle.

Lemma 10. *Let a, b be any two distinct vertices of a digraph* $G = (V, A)$ *that satisfy Property 1. Then G is* 2*-vertex connected.*

Proof. In light of Lemma 4 it suffices to show that for any pair $c, d \in V$, G contains the arc (d, c) or two internally vertex-disjoint paths from d to c. This follows immediately from Lemma 3 when $d \in \{a, b\}$ or $c \in \{a, b\}$. Now consider $d \notin \{a, b\}$ and $c \notin \{a, b\}$. We will exhibit two internally vertex-disjoint paths from d to c. By Lemma 9 we have two internally vertex-disjoint paths $Q[d, a]$ and $Q'[d, b]$, i.e., $Q(d, a] \cap Q'(d, b] = \emptyset$. In particular, note that $a \notin Q'[d, b]$ and $b \notin Q[d, a]$. We distinguish the following cases:

a) Suppose $(a, c) \in A$. First consider that also $(b, c) \in A$. If none of the paths $Q[d, a]$ and $Q'[d, b]$ contains c then $Q[d, a] \cdot (a, c)$ and $Q'[d, b] \cdot (b, c)$ are internally vertex-disjoint. If $Q[d, a]$ contains c then $Q'[d, b]$ does not, so $Q[d, c]$ and $Q'[d, b] \cdot (b, c)$ are internally vertex-disjoint. The case $c \in Q'[d, b]$ is symmetric.

If $(b, c) \notin A$ then there are two internally vertex-disjoint paths $P_1[b, c]$ and $P_2[b, c]$. Suppose $Q[d, a] \cap P_1[b, c] = \emptyset$. Let $R = Q[d, a] \cdot (a, c)$ and $R' = Q'[d, b] \cdot P_1[b, c]$. Then $R[d, c] (= R)$ and $R'[d, c]$ are internally vertex-disjoint. Now consider $Q[d, a] \cap P_1[b, c] \neq \emptyset$. We would like to apply Lemma 5 for vertices d, b and c but we need two internally vertex-disjoint paths from d to b. To that end, let us assume that $(a, b) \in A$ and set $Q''[d, b] = Q[d, a] \cdot (a, b)$. Then Lemma 5 provides two internally vertex-disjoint paths $R[d, c]$ and $R'[d, c]$. If none of these paths uses the arc (a, b) then we have found two internally vertex-disjoint paths

from d to c in G. Otherwise, suppose that $R[d,c]$ contains (a,b). Then the paths $R[d,a] \cdot (a,c)$ and $R'[d,c]$ are internally vertex-disjoint.

The case $(b,c) \in A$ is symmetric.

b) The case $(d,a) \in A$ can be analyzed similarly to case (a) but we provide the details for completeness. From Lemma 7 we have two internally vertex-disjoint paths $Q[a,c]$ and $Q'[b,c]$, i.e., $Q[a,c) \cap Q'[b,c) = \emptyset$. In particular, note that $a \notin Q'[b,c]$ and $b \notin Q[a,c]$. First consider that also $(d,b) \in A$. If none of the paths $Q[a,c]$ and $Q'[b,c]$ contains d then $(d,a) \cdot Q[a,c]$ and $(d,b) \cdot Q'[b,c]$ are internally vertex-disjoint. If $Q[a,c]$ contains d then $Q'[b,c]$ does not, so $Q[d,c]$ and $(d,b) \cdot Q'[b,c]$ are internally vertex-disjoint. The case $d \in Q'[b,c]$ is symmetric.

If $(d,b) \notin A$ then there are two internally vertex-disjoint paths $P_1[d,b]$ and $P_2[d,b]$. Suppose $Q[a,c] \cap P_1[d,b] = \emptyset$. Let $R = (d,a) \cdot Q[a,c]$ and $R' = P_1[d,b] \cdot Q'[b,c]$. Then $R[d,c] (= R)$ and $R'[d,c]$ are internally vertex-disjoint. Now consider $Q[a,c] \cap P_1[d,b] \neq \emptyset$. We apply Lemma 5 for vertices d, b and c as in case (a). We assume at first that $(b,a) \in A$ and set $Q''[b,c] = (b,a) \cdot Q[a,c]$. Then Lemma 5 gives us two internally vertex-disjoint paths $R[d,c]$ and $R'[d,c]$. If none of these paths uses the arc (b,a) then we have found two internally vertex-disjoint paths from d to c in G. Otherwise, suppose that $R[d,c]$ contains (b,a). Then the paths $(d,a) \cdot R[a,c]$ and $R'[d,c]$ are internally vertex-disjoint.

The case $(d,b) \in A$ is symmetric.

c) It remains to examine the case where none of the arcs (a,c), (d,a), (b,c) and (d,b) is present. By Lemma 3 we have the following pairs of internally vertex-disjoint paths: $P_1[d,a]$ and $P_2[d,a]$, $P_3[a,c]$ and $P_4[a,c]$, $P_5[d,b]$ and $P_6[d,b]$, and $P_7[b,c]$ and $P_8[b,c]$. (See Figure 2(b) in Appendix A.) If $P_1[d,a)$ or $P_2[d,a)$ intersects $P_3(a,c]$ or $P_4(a,c]$ then Lemma 5 gives us two internally vertex-disjoint paths from d to c. Similarly, if $P_5[d,b)$ or $P_6[d,b)$ intersects $P_7(b,c]$ or $P_8(b,c]$ then Lemma 5 gives us two internally vertex-disjoint paths from d to c. Now we suppose that $P_1[d,a)$ and $P_2[d,a)$ do not intersect $P_3(a,c]$ and $P_4(a,c]$, and also that $P_5[d,b)$ and $P_6[d,b)$ do not intersect $P_7(b,c]$ and $P_8(b,c]$.

First we consider that $P_5[d,b]$ intersects $P_3[a,c] \cup P_4[a,c]$. Let f be the vertex with the lowest rank in $P_5[d,b]$ such that $f \in P_5[d,b] \cap \big(P_3[a,c] \cup P_4[a,c]\big)$. Without loss of generality we can assume that $f \in P_3[a,c]$. Suppose $f = a$. Then $a \notin P_6[d,b]$. If $P_6[d,b]$ intersects $P_3[a,c] \cup P_4[a,c]$ then we can alternate the role of $P_5[d,b]$ and $P_6[d,b]$ and consider the case $f \neq a$. So now let $P_6[d,b] \cap \big(P_3[a,c] \cup P_4[a,c]\big) = \emptyset$. Let h be the vertex with the lowest rank in $P_7[b,c]$ such that $h \in P_7[b,c] \cap \big(P_3[a,c] \cup P_4[a,c]\big)$. Without loss of generality, suppose $h \in P_3[a,c]$. Then the paths $P_5[d,a] \cdot P_4[a,c]$ and $P_6[d,b] \cdot P_7[b,h] \cdot P_3[h,c]$ are internally vertex-disjoint. Next, consider $f \neq a$. Let e be the vertex with the highest rank in $P_5[d,f]$ such that $e \in P_5[d,f] \cap \big(P_1[d,a] \cup P_2[d,a]\big)$. Note that $e \neq a$. (Also $e \neq f$). Without loss of generality, suppose $e \in P_1[d,a]$. Then $P_1[d,e] \cdot P_5[e,f] \cdot P_3[f,c]$ and $P_2[d,a] \cdot P_4[a,c]$ are internally vertex-disjoint.

The cases $P_6[d,b] \cap \big(P_3[a,c] \cup P_4[a,c]\big) \neq \emptyset$, $P_1[d,a] \cap \big(P_7[b,c] \cup P_8[b,c]\big) \neq \emptyset$, and $P_2[d,a] \cap \big(P_7[b,c] \cup P_8[b,c]\big) \neq \emptyset$ are treated similarly.

Finally suppose that $P_5[d,b]$ and $P_6[d,b]$ do not intersect $P_3[a,c]$ and $P_4[a,c]$, and also that $P_1[d,a]$ and $P_2[d,a]$ do not intersect $P_7[b,c]$ and $P_8[b,c]$. We apply

Lemma 8 for a, b, and d and get two internally vertex-disjoint paths $Q_1[d,a]$ and $Q_2[d,b]$. Then we apply Lemma 6 for a, b, and c get two internally vertex-disjoint paths $Q_3[a,c]$ and $Q_4[b,c]$. Since $\big(P_1[d,a] \cup P_2[d,a]\big) \cap \big(P_7[b,c] \cup P_8[b,c]\big) = \emptyset$ and $\big(P_5[d,b] \cup P_6[d,b]\big) \cap \big(P_3[a,c] \cup P_4[a,c]\big) = \emptyset$, we have $Q_1[d,a] \cap \big(Q_3(a,c] \cup Q_4[b,c]\big) = \emptyset$ and $Q_2[d,b] \cap \big(Q_3[a,c] \cup Q_4(b,c]\big) = \emptyset$.Thus $Q_1[d,a] \cdot Q_3[a,c]$ and $Q_2[d,b] \cdot Q_4[b,c]$ are internally vertex-disjoint. □

Combining Lemmas 2 and 10 we have:

Theorem 1. *Let a, b be two arbitrary but distinct vertices of a digraph G. Then G is 2-vertex connected if and only if it satisfies Property 1 for a and b.*

2.2 Linear-time algorithm

Based on Theorem 1 we propose the following algorithm for testing 2-vertex connectivity: Given the input digraph $G = (V, A)$, we first compute the reverse graph $G^r = (V, A^r)$, where $A^r = \{(x, y) \mid (y, x) \in A\}$. Then we pick two distinct vertices $a, b \in V$ and verify that for each $s \in \{a, b\}$ the flowgraphs $G(s)$ and $G^r(s)$ have trivial dominators only; we report that G is 2-vertex connected if and only if this is true. If the input graph is strongly connected but not 2-vertex connected, then we would like to report a cut vertex, i.e., a vertex whose removal increases the number of strongly connected components. To that end, we can use another property of the trivial-dominator-verification algorithm in [16]. Namely, if a flowgraph $G(s)$ has non-trivial dominators then the verification algorithm reports two vertices $x \neq s$ and y, such that x is the immediate dominator of y in $G(s)$. Thus, the removal of x destroys all paths from s to y.

The correctness of the above algorithm follows immediately from Theorem 1. We now turn to the running time. Computing G^r in $O(m+n)$ time is easy. Furthermore, we can test if a flowgraph has only trivial dominators in $O(m+n)$ time using the algorithm [16]. (Alternatively, we can use a linear-time algorithm for computing the dominators of the flowgraph but this computation is more complicated [5].) Since our 2-vertex-connectivity algorithm uses four such tests, the total running time is $O(m+n)$. In practice, we can use a simpler $O(m\alpha(m,n))$-time version of the trivial-dominator-verification algorithm in [16], which uses a standard disjoint set union data structure [13] instead of the linear-time algorithm of Gabow and Tarjan [13].

Theorem 2. *There is a linear-time algorithm for testing 2-vertex connectivity of a digraph G. If G is strongly connected but not 2-vertex connected then the algorithm returns a cut vertex.*

3 Computing two vertex-disjoint *s*-*t* paths

We now consider the problem of preprocessing a 2-vertex connected digraph $G = (V, A)$ into a data structure that can efficiently compute two internally

vertex-disjoint paths from d to c, for any pair of distinct vertices $d, c \in V$. Our data structure is based on the proof of Theorem 1 and on a linear-time algorithm of [16], which computes two branchings T_1 and T_2 of a flowgraph $G(s) = (V, A, s)$. These are (directed) spanning trees of G, which are rooted at s and have the following property: For any vertex $v \in V$, the two directed paths from s to v in T_1 and T_2, denoted by $T_1[s, v]$ and $T_2[s, v]$ respectively, meet only at the dominators of v in $G(s)$. Therefore, if $G = (V, A)$ is 2-vertex connected then, by Lemma 2, $T_1[s, v]$ and $T_2[s, v]$ are internally vertex-disjoint; two branchings that have this property are called *independent*. We begin by computing the following pairs of independent branchings: T_1 and T_2 of $G^r(a)$, T_3 and T_4 of $G(a)$, T_5 and T_6 of $G^r(b)$, and T_7 and T_8 of $G(b)$, where a and b are two arbitrary but distinct vertices of G. Let A' be the set of arcs in these eight trees. Then, Theorem 1 implies that the digraph $G' = (V, A')$ is a 2-vertex connected spanning subgraph of G. Therefore, we can compute a pair of internally vertex-disjoint d-c paths in G'. This computation takes $O(n)$ time with a flow-augmenting algorithm (see, e.g., [3]), since A' has $O(n)$ arcs.

Next, we describe how to compute these paths in $O(\log^2 n)$ time, so that we can report them in additional $O(k)$ time, where k is the total length of the two paths. The proof of Theorem 1 finds two internally vertex-disjoint paths from d to c, using the following four pairs of internally vertex-disjoint paths: $P_1[d, a]$ and $P_2[d, a]$, $P_3[a, c]$ and $P_4[a, c]$, $P_5[d, b]$ and $P_6[d, b]$, and $P_7[b, c]$ and $P_8[b, c]$. In order to answer a query for two internally vertex-disjoint paths from d to c, we can use the corresponding paths on the branchings $T_1, \ldots, T_8$, i.e., we set $P_1[d, a] = (T_1[a, d])^r$, $P_2[d, a] = (T_2[a, d])^r$, $P_3[a, c] = T_3[a, c]$, $P_4[a, c] = T_4[a, c]$, $P_5[d, b] = (T_5[b, d])^r$, $P_6[d, a] = (T_6[a, d])^r$, $P_7[a, c] = T_7[a, c]$, $P_8[a, c] = T_8[a, c]$.

Let S_1 and S_2 be any two rooted trees on the same set of vertices. We define a set of operations on S_1 and S_2 that enable an efficient implementation of the construction given in Section 2.1. Consider four vertices x_1, y_1, x_2 and y_2, such that x_1 is an ancestor of y_1 in S_1 and x_2 is an ancestor of y_2 in S_2. We need a data structure that supports the following set of operations:

(i) Test if $S_1[x_1, y_1]$ contains x_2.
(ii) Return the topmost vertex in $S_1(x_1, y_1]$.
(iii) Test if $S_1[x_1, y_1]$ and $S_2[x_2, y_2]$ contain a common vertex.
(iv) Find the lowest ancestor of y_2 in $S_2[x_2, y_2]$ that is contained in $S_1[x_1, y_1]$.
(v) Find the highest ancestor of y_2 in $S_2[x_2, y_2]$ that is contained in $S_1[x_1, y_1]$.

By examining the construction of Section 2.1 we can verify that the above operations suffice for our needs. (We need a constant number of these operations per query.) For instance, we can find the vertex e with the highest rank in $P_5[d, f]$ such that $e \in P_5[d, f] \cap P_1[d, a]$ (refer to case (c) in the proof of Lemma 10) by applying operation (v) with $S_1 = T_1$, $S_2 = T_5$, $x_1 = a$, $y_1 = d$, $x_2 = f$ and $y_2 = d$. Operation (ii) is useful when we want to exclude the first vertex of a path in some computation. Excluding the last vertex on a path of S_j ($j \in \{1, 2\}$) is straightforward if we maintain a pointer from a node to its parent in S_j.

We develop an $O(n)$-space data structure that supports operations (i)-(v) efficiently. First note that operation (i) can be answered from (iii) if we set $y_2 =$

x_2. Furthermore, operation (iii) can be answered from (iv) or (v); if $S_1[x_1, y_1] \cap S_2[x_2, y_2] = \emptyset$ then these operations return null. Now it remains to implement operations (ii), (iv) and (v). We begin by assigning a depth-first search interval (as in [11]) to each vertex in S_1. Let $I_1(x) = [s_1(x), t_1(x)]$ be the interval of a vertex x in S_1; $s_1(x)$ is the time of the first visit to x (during the depth-first search) and $t_1(x)$ is the time of the last visit to x. (These times are computed by incrementing a counter after visiting or leaving a vertex during the search.) This way all the assigned $s_1()$ and $t_1()$ values are distinct, and for any vertex x we have $1 \leq s_1(x) < t_1(x) \leq 2n$. Moreover, by well-known properties of depth-first search, we have that x is an ancestor of y in S_1 if and only if $I_1(y) \subseteq I(x)$; if x and y are unrelated in S_1 then $I_1(x)$ and $I_1(y)$ are disjoint. Now, for operation (ii) we simply need to locate the child z of x_1 in S_1 such that $s_1(y_1) \in I_1(z)$. This is a static predecessor search problem that can be solved in $O(\log n)$ time with binary search (which suffices here) or in $O(\log \log n)$ time with more advanced structures [4].

In order to support operations (iii)-(v) efficiently we also assign a depth-first search interval $I_2(x) = [s_2(x), t_2(x)]$ to each vertex x in S_2. Next, we map each vertex x to an axis-parallel rectangle $R(x) = I_1(x) \times I_2(x)$. (See Figure 3 in Appendix A.) Let $\mathcal{R}$ be the set of all axis-parallel rectangles $R(x)$. We implement operations (iv) and (v) as ray shooting queries in the subdivision induced by $\mathcal{R}$. For any two vertices x and y, we define $R(x, y) = I_1(x) \times I_2(y)$. Then, $R(x) \equiv R(x, x)$. Consider two rectangles $R(x_1, x_2)$ and $R(y_1, y_2)$. If $I_1(x_1) \cap I_1(y_1) = \emptyset$ or $I_2(x_2) \cap I_2(y_2) = \emptyset$, then $R(x_1, x_2)$ and $R(y_1, y_2)$ do not intersect. Now suppose that both $I_1(x_1) \cap I_1(y_1) \neq \emptyset$ and $I_2(x_2) \cap I_2(y_2) \neq \emptyset$. Let $I_1(y_1) \subseteq I_1(x_1)$. If also $I_2(y_2) \subseteq I_2(x_2)$ then $R(y_1, y_2)$ is contained in $R(x_1, x_2)$; we denote this by $R(y_1, y_2) \subseteq R(x_1, x_2)$. Otherwise, if $I_2(x_2) \subseteq I_2(y_2)$ then both vertical edges of $R(y_1, y_2)$ intersect both horizontal edges of $R(x_1, x_2)$. Next, consider a rectangle $R(x_1, x_2)$ and let $\mathcal{R}(x_1, x_2) = \{R(z) \in \mathcal{R} \ : \ R(x_1, x_2) \subseteq R(z)\}$, i.e., the rectangles in $\mathcal{R}$ containing $R(x_1, x_2)$. The properties of the intervals $I_1()$ and $I_2()$ imply that we can order the rectangles in $\mathcal{R}(x_1, x_2)$ with respect to their vertical distance from $R(x_1, x_2)$.[2] More formally, let $R(z_1), R(z_2), \ldots, R(z_\xi)$ be the rectangles in $\mathcal{R}(x_1, x_2)$ ordered by increasing $t_2(z_j)$ (the height of the upper horizontal edge). Also, let $R(z_{i_1}), R(z_{i_2}), \ldots, R(z_{i_\xi})$ be the rectangles in $\mathcal{R}(x_1, x_2)$ ordered by decreasing $s_2(z_{i_j})$ (the height of the lower horizontal edge). Then $i_j = j$, $j = 1, \ldots, \xi$. Now let $Q = R(y_1, y_2)$ and let $Q' = R(x_1, x_2)$. To perform operation (iv) we locate the rectangle $R(z) \in \mathcal{R}(y_1, y_2)$ with minimum $t_2(z)$ (and maximum $s_2(z)$). For operation (v) we locate the rectangle $R(z) \in \mathcal{R}(y_1, y_2)$ with maximum $t_2(z)$ (and minimum $s_2(z)$) such that $R(z) \subseteq Q'$. (See Figure 3 in Appendix A.) We can perform these operations efficiently by adapting a data structure of Chazelle [8]. The data structure consists of a binary search tree T on the vertical coordinates $s_2()$ and $t_2()$ of the vertices, and is constructed as follows. Let ℓ be an infinite horizontal line with $|\mathcal{R}|$ horizontal rectangle edges on each side. This line partitions $\mathcal{R}$ into three subsets: $\mathcal{R}_\uparrow$ contains the rectangles

[2] The same holds for the horizontal distance, but we only need the vertical distance for the operations as defined.

completely above ℓ, $\mathcal{R}_\downarrow$ contains the rectangles completely bellow ℓ, and $\mathcal{R}_\ell$ contains the rectangles intersecting ℓ. We associate with the root r of T the set $\mathcal{R}_\ell$. The left (resp. right) subtree of r is defined recursively for the set $\mathcal{R}_\downarrow$ (resp. $\mathcal{R}_\uparrow$). Clearly, T has $O(\log n)$ height. For any node $v \in T$, we let $\mathcal{R}(v)$ denote the set of rectangles associated with v.

Let $q = (s_1(y_1), s_2(y_2))$, i.e., the lower left corner of Q. (Any corner of Q will do as well.) We implement operation (iv) as a ray shooting query in the subdivision induced by $\mathcal{R}(v_i)$, for each node $v_i \in T$ on the path $(v_0, v_1, \ldots, v_h)$ from $r = v_0$ to the leaf v_h that corresponds to the vertical coordinate $s_2(y_2)$. Suppose that the horizontal line $\ell(v_i)$ associated with node v_i is above q. Then, we locate the first rectangle $R_q^i \in \mathcal{R}(v_i)$ that is intersected by the vertical ray $[q, (s_1(y_1), -\infty)]$. If $\ell(v_i)$ is below q then we locate the first rectangle $R_q^i \in \mathcal{R}(v_i)$ that is intersected by the vertical ray $[q, (s_1(y_1), +\infty)]$. In either case, R_q^i can be found in $O(\log n)$ time using a planar point location data structure [22]. The answer to query (iv) is the rectangle $R(z) \in \{R_q^0, R_q^1, \ldots, R_q^h\}$ with minimum $t_2(z)$, therefore it can be found in total $O(h \log n) = O(\log^2 n)$ time. Operation (v) is carried out similarly. Let $q'_s = (s_1(y_1), s_2(x_2))$ and $q'_t = (s_1(y_1), t_2(x_2))$, respectively, be the projection of q on the lower and upper edge of Q'. Again we perform a ray shooting query in the subdivision induced by $\mathcal{R}(v_i)$, for each node $v_i \in T$ on the path $(v_0, v_1, \ldots, v_h)$ from $r = v_0$ to the leaf v_h that corresponds to the vertical coordinate $s_2(y_2)$. Suppose that the horizontal line $\ell(v_i)$ associated with node v_i is above q. Then, we locate the first rectangle $R_q^i \in \mathcal{R}(v_i)$ that is intersected by the vertical ray $[q'_s, (s_1(y_1), +\infty)]$. If $\ell(v_i)$ is below q then we locate the first rectangle $R_q^i \in \mathcal{R}(v_i)$ that is intersected by the vertical ray $[q'_t, (s_1(y_1), -\infty)]$. The answer to query (v) is the rectangle $R(z) \in \{R_q^0, R_q^1, \ldots, R_q^h\}$ with maximum $t_2(z)$ such that $Q \subseteq R(z) \subseteq Q'$. Therefore, it can be found in total $O(\log^2 n)$ time. It is easy to verify that the space bound for the above data structure is $O(n)$. For the construction of Section 2.1 we actually need such a data structure for several pairs of the branchings $T_1, \ldots, T_8$, but the total space is still $O(n)$.

Theorem 3. *Let $G = (V, A)$ be a 2-vertex connected digraph $G = (V, A)$ with n vertices. We can construct an $O(n)$-space data structure that can compute two internally vertex-disjoint paths from d to c in $O(\log^2 n)$ time, for any two distinct vertices $d, c \in V$. The two paths can be reported in additional $O(k)$ time, where k is their total length.*

We remark that the query time can be reduced to $O(\log n \sqrt{\log n / \log\log n})$ by applying the result of [7]. We leave the design of more efficient structures for the operations (i)-(v) as an open problem.

References

1. A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, Reading, MA, 1986.
2. S. Alstrup, D. Harel, P. W. Lauridsen, and M. Thorup. Dominators in linear time. *SIAM Journal on Computing*, 28(6):2117–32, 1999.

3. J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications (Springer Monographs in Mathematics)*. Springer, 1st ed. 2001. 3rd printing edition, 2002.
4. P. Beame and F. E. Fich. Optimal bounds for the predecessor problem and related problems. *J. Comput. Syst. Sci.*, 65(1):38–72, 2002.
5. 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(4):1533–1573, 2008.
6. A. L. Buchsbaum, H. Kaplan, A. Rogers, and J. R. Westbrook. A new, simpler linear-time dominators algorithm. *ACM Transactions on Programming Languages and Systems*, 20(6):1265–96, 1998. Corrigendum appeared in 27(3):383-7, 2005.
7. T. M. Chan and M. Patraşcu. Transdichotomous results in computational geometry, i: Point location in sublogarithmic time. *SIAM Journal on Computing*, 39(2):703–729, 2009.
8. B. Chazelle. Filtering search: A new approach to query-answering. *SIAM Journal on Computing*, 15(3):703–24, 1986.
9. J. Cheriyan and J. H. Reif. Directed *s-t* numberings, rubber bands, and testing digraph *k*-vertex connectivity. *Combinatorica*, pages 435–451, 1994.
10. D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.
11. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.
12. H. N. Gabow. Using expander graphs to find vertex connectivity. *Journal of the ACM*, 53(5):800–844, 2006.
13. H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30(2):209–21, 1985.
14. L. Georgiadis. *Linear-Time Algorithms for Dominators and Related Problems*. PhD thesis, Princeton University, 2005.
15. L. Georgiadis and R. E. Tarjan. Finding dominators revisited. In *Proc. 15th ACM-SIAM Symp. on Discrete Algorithms*, pages 862–871, 2004.
16. L. Georgiadis and R. E. Tarjan. Dominator tree verification and vertex-disjoint paths. In *Proc. 16th ACM-SIAM Symp. on Discrete Algorithms*, pages 433–442, 2005.
17. L. Georgiadis, R. E. Tarjan, and R. F. Werneck. Finding dominators in practice. *Journal of Graph Algorithms and Applications (JGAA)*, 10(1):69–94, 2006.
18. M. R. Henzinger, S. Rao, and H. N. Gabow. Computing vertex connectivity: New bounds from old techniques. *Journal of Algorithms*, 34:222–250, 2000.
19. J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973.
20. T. Lengauer and R. E. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Transactions on Programming Languages and Systems*, 1(1):121–41, 1979.
21. H. Nagamochi and T. Ibaraki. A linear-time algorithm for finding a sparse *k*-connected spanning subgraph of a *k*-connected graph. *Algorithmica*, 7:583–596.
22. N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29(7):669–679, 1986.
23. R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–59, 1972.
24. R. E. Tarjan. Edge-disjoint spanning trees and depth-first search. *Acta Informatica*, 6(2):171–85, 1976.

A Omitted Figures

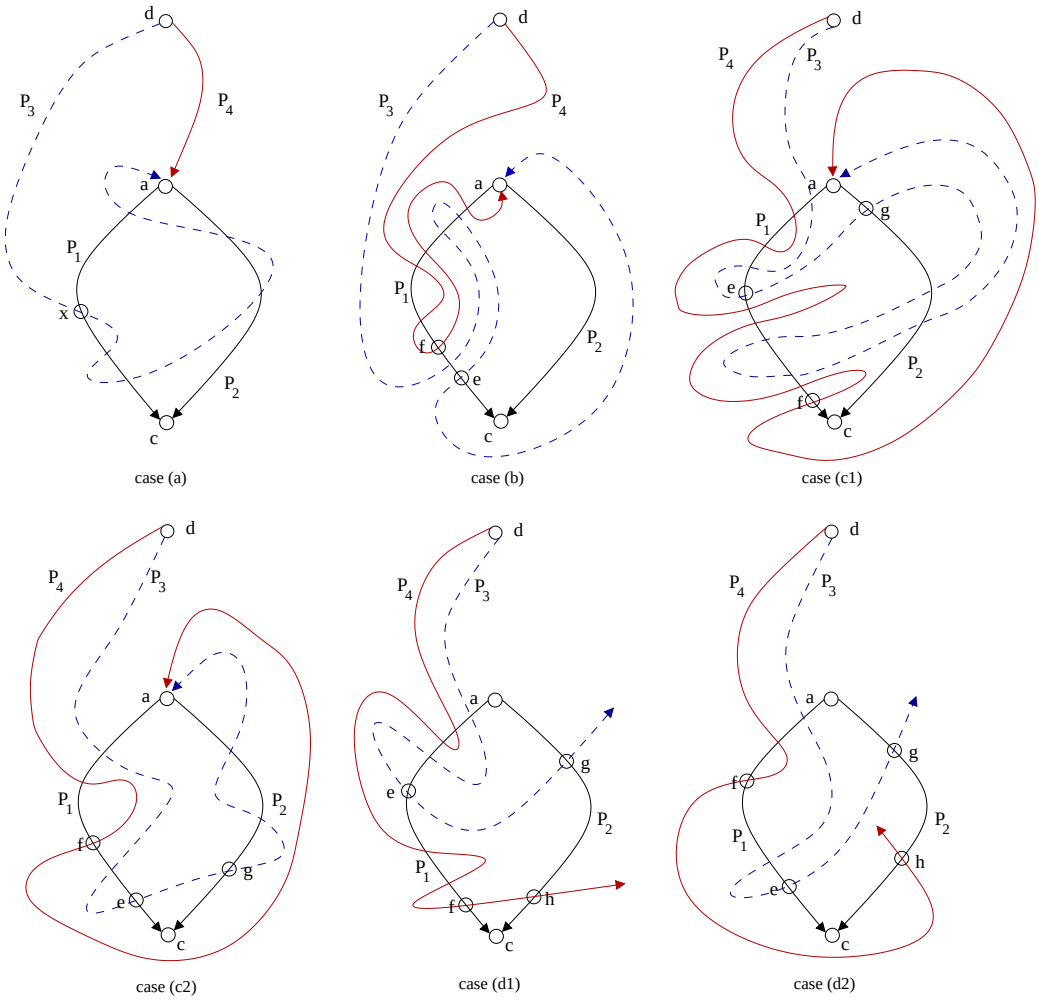

Fig. 1. Cases considered in the proof of Lemma 5.

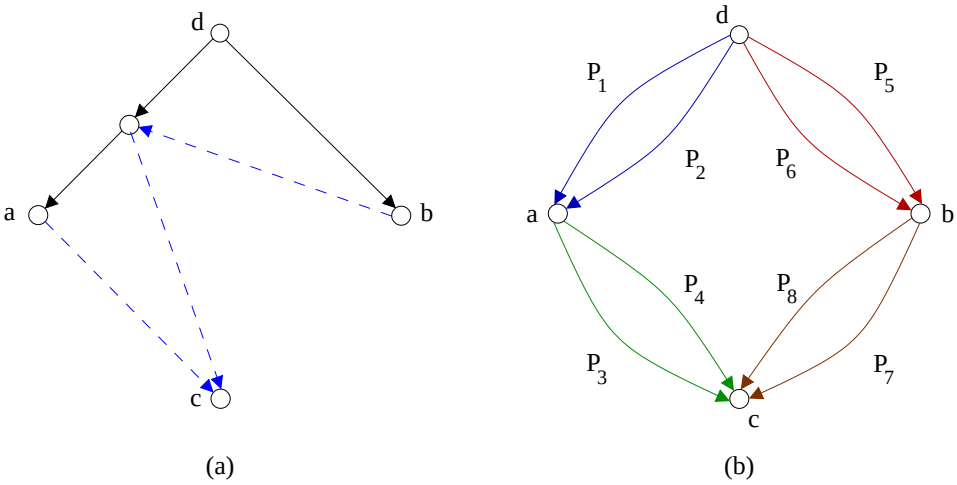

Fig. 2. (a) An example where a pair of internally vertex-disjoint paths from d to a and from d to b (solid arcs) and a pair of internally vertex-disjoint paths from a to c and from b to c (dashed arcs) do not suffice to give us a pair of internally vertex-disjoint paths from d to c. (b) Paths used in the proof of case (c) of Lemma 10.

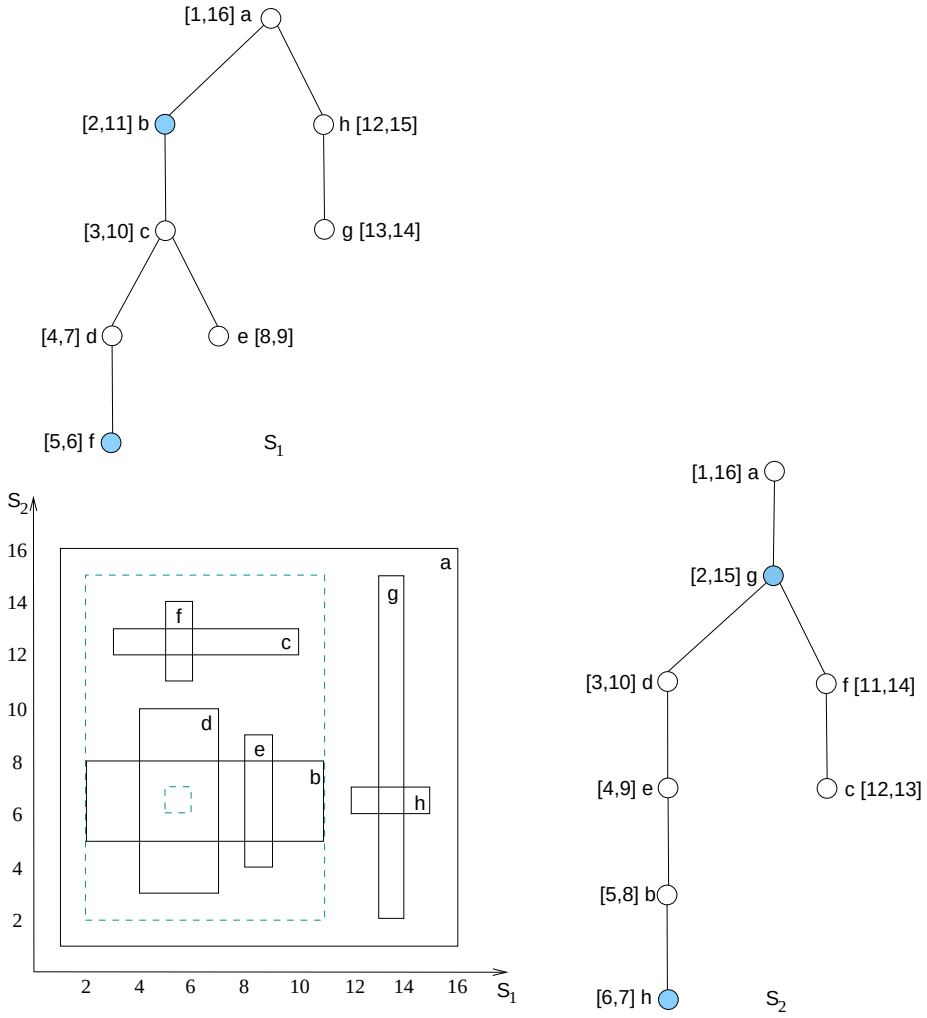

Fig. 3. Answering queries of type (iv) and (v). In this example $x_1 = b$, $y_1 = f$, $x_2 = g$ and $y_2 = h$. The small dashed rectangle is $Q = R(y_1, y_2)$, and the large dashed rectangle is $Q' = R(x_1, x_2)$. The answer to queries (iv) and (v) are, respectively, vertices b and d.