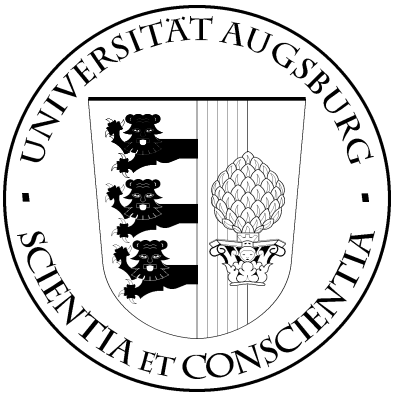
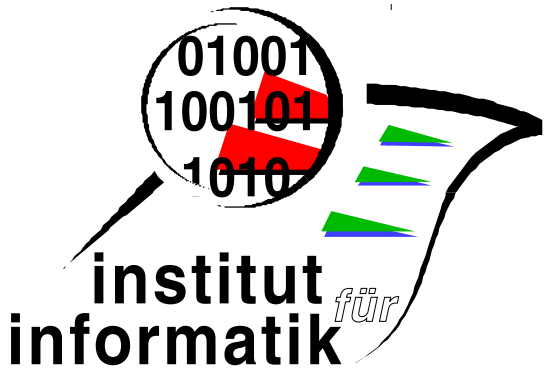
Universität Augsburg




# Finding Disjoint Paths on Directed Acyclic Graphs

**Torsten Tholey**

Report 2005-04 Februar 2005




Institut für Informatik
D-86135 Augsburg

# Finding Disjoint Paths on Directed Acyclic Graphs

Torsten Tholey

Institut für Informatik
Universität Augsburg
D-86135 Augsburg, Germany
tholey@informatik.uni-augsburg.de

**Abstract.** Given $k+1$ pairs of vertices $(s_1, s_2), (u_1, v_1), \ldots, (u_k, v_k)$ of a directed acyclic graph, we show that a modified version of a data structure of Suurballe and Tarjan can output, for each pair $(u_l, v_l)$ with $1 \leq l \leq k$, a tuple $(s_1, t_1, s_2, t_2)$ with $\{t_1, t_2\} = \{u_l, v_l\}$ in constant time such that there are two disjoint paths $p_1$, from $s_1$ to $t_1$, and $p_2$, from $s_2$ to $t_2$, if such a tuple exists. Disjoint can mean vertex- as well as edge-disjoint. As an application we show that the presented data structure can be used to improve the previous best known running time $O(mn)$ for the so called 2-disjoint paths problem on directed acyclic graphs to $O(m \log_{2+m/n} n + n \log^3 n)$. In this problem, given four vertices $s_1, s_2, t_1$, and $t_2$, we want to construct two disjoint paths $p_1$, from $s_1$ to $t_1$, and $p_2$, from $s_2$ to $t_2$, if such paths exist.

## 1 Introduction

The problem of finding disjoint paths is one of the fundamental problems in graph theory with many applications concerning network reliability, routing problems, VLSI-design, ... Such problems have been studied extensively and a variety of efficient algorithm are known for undirected graphs (cf. the overviews given in [2] and [3]), whereas much less is known about finding disjoint paths on directed graphs.

*Previous results:* Given $2k$ vertices $s_1, \ldots, s_k$, $t_1, \ldots, t_k$, one simple path finding problem consists of determining $k$ disjoint paths $p_i$ ($i \in \{1, \ldots, k\}$) between the vertices $\{s_1, \ldots, s_k\}$ and $\{t_1, \ldots, t_k\}$ with $p_i$ leading from $s_i$ to $t_{\pi(i)}$ for a permutation $\pi$ of the numbers $1, \ldots, k$. This problem can be solved with standard network flow techniques for directed as well as for undirected graphs and for both, vertex- and edge-disjoint disjoint paths. For fixed $k \in \mathbb{N}$, this leads to a running time of $O(m+n)$, where here and in the following $m$ will denote the number of edges and $n$ the number of vertices of the graph under consideration.

For undirected graphs and $k \in \{2, 3\}$, Di Battista, Tamassia, and Vismara [2] have shown that allowing a preprocessing time of $O(m+n)$ one can test the existence of $k$ vertex-disjoint paths between two vertices in constant time and output $k$ such paths, if they exist, in a time linear in the number of the edges visited by these paths. Di Battista, Tamassia, and Vismara also gave an overview over other data structures supporting the above queries for $k \geq 4$. For results concerning edge-disjoint paths between pairs of vertices we refer the reader to the paper of Dinitz and Westbrook [3].

For a directed graph $G = (V, E)$ and a fixed vertex $s \in V$, Suurballe and Tarjan [14] presented a data structure with a preprocessing time of $O(n + m \log_{2+m/n} n)$ which, for each $t \in V$, can test in constant time whether there are two disjoint paths from $s$ to $t$, and, if so, can output such paths in linear time. The result holds for both, vertex- and edge-disjoint paths. Unfortunately, no generalization of this data structure is known to test the existence of $k \geq 3$ disjoint paths, and the best known data structure for finding two disjoint paths between arbitrary pairs of vertices, i.e. $s$ not fixed, is to construct, for each vertex $s \in V$ as source

vertex, a separate data structure of the kind described by Suurballe and Tarjan. The data structure of Suurballe and Tarjan can also be used to find two vertex- or edge-disjoint paths from one fixed vertex $s$ to two different vertices $t_1, t_2$ with $(t_1, t_2)$ being equal to one of $k$ fixed pairs of vertices $(u_1, v_1), \ldots, (u_k, v_k)$ : Just add $k$ extra vertices $w_1, \ldots, w_k$ and $2k$ extra edges $(u_1, w_1), (v_1, w_1), \ldots, (u_k, w_k), (v_k, w_k)$ to $G$. As a consequence, the preprocessing time increases to $O(n + (m + k) \log_{2+(m+k)/(n+k)} n)$, but the time needed to test the existence of disjoint paths will remain constant, and two such paths, if they exist, can be output in linear time.

Another interesting paths finding problem is the $k$-disjoint paths problem. In this problem we are given $2k$ vertices $s_1, \ldots, s_k, t_1, \ldots, t_k$ and we want to construct $k$ disjoint paths $p_i$ $(1 \leq i \leq k)$, from $s_i$ to $t_i$. For short, we will refer to this problem as the $k$-DPP or, more precisely, as $k$-VDPP, if disjoint means vertex-disjoint, and as $k$-EDPP, if disjoint means edge disjoint.

The first polynomial time algorithms for the $k$-VDPP where given by Ohtsuki [7], Seymour [12], Shiloach [13], and Thomassen [17], for $k = 2$, and by Robertson and Seymour [10] for general $k$. Some papers only consider the decision version of the $k$-VDPP, where the existence of disjoint paths is being tested without computing them explicitely. However, there is a polynomial time reduction from this version of the problem to the general $k$-VDPP (cf. [16]). With the line-graph reduction described by Perl and Shiloach in [9] the $k$-EDPP can be also solved in polynomial time. If we let $\alpha$ be the inverse Ackerman function as defined in [15], the currently best known time bounds for the $k$-DPP, are $O(n + m\alpha(m, n))$ for the 2-VDPP, $O(n \log n + m\alpha(m, n))$ time for the 2-EDPP as shown by the author of this paper in [16], and $O(mn^2)$ time for the $k$-VDPP with $k > 3$, and $O(m^2n^2)$ for the $k$-EDPP with $k > 3$ as shown by Perković and Reed in [8].[1]

For directed graphs, the decision versions of the $k$-EDPP and the $k$-VDPP are $\mathcal{NP}$-complete, even for $k = 2$, as shown by Fortune, Hopcroft, and Wyllie [4]. However, in [9] Perl and Shiloach presented an $O(mn)$-time algorithm for solving the 2-VDPP and the 2-EDPP on dags (directed acyclic graphs). Fortune, Hopcroft, and Wyllie [4] generalized this result of Perl and Shiloach to a polynomial-time algorithm for the $k$-VDPP on dags for all $k \in \mathbb{N}$. Lucchesi and Giglio [6] described a linear time reduction from the decision version of the 2-VDPP on dags to the decision version of the 2-VDPP on undirected graphs. They also presented a corollary from which it follows that there is always a solution of the 2-VDPP on the undirected graph after the reduction, if this graph is non-planar. Since Perl and Shiloach [9] have shown that the 2-VDPP on undirected planar graphs is solvable in linear time, the decision version of the 2-VDPP on dags is also solvable in linear time. Finally, applying the reduction from the 2-EDPP on dags to the 2-VDPP on dags given in [16] there is an $O(n + m \log_{2+m/n} n)$ time algorithm for solving the decision version of the 2-EDPP on dags. As an application of the $k$-EDPP on dags, Schrijver [11] described an airplane routing problem that can be solved with an algorithm for the $k$-EDPP on dags.

*New results.* In some scenarios, given four vertices $s_1, s_2, t_1$, and $t_2$, apart from testing whether there are two disjoint paths leading from the vertices in $\{s_1, s_2\}$ to the vertices in $\{t_1, t_2\}$ we might also be interested in knowing whether the path starting in $s_1$ leads to $t_1$ or $t_2$. Given $k + 1$ pairs of vertices $(s_1, s_2), (u_1, v_1), \ldots, (u_k, v_k)$ of a directed graph, we present in Section 3 a modified version of a data structure of Suurballe and Tarjan which can output, for each pair $(u_l, v_l)$ with $1 \leq l \leq k$, a tuple $(s_1, t_1, s_2, t_2)$ with $\{t_1, t_2\} = \{u_l, v_l\}$ in constant time

[1] For the last two results we also use the line graph reduction from the $k$-EDPP to the $k$-VDPP as well as the reduction from the decision version to the general version of the $k$-DPP.

such that there are two vertex- or, alternatively, edge-disjoint paths $p_1$, from $s_1$ to $t_1$, and $p_2$, from $s_2$ to $t_2$. This data structure can be constructed in $O(n \log^2 n + (m+k) \log_{2+(m+k)/(n+k)} n)$ time.

As an application of this data structure and main result of this paper, we show that it can be used to improve the running time for the 2-VDPP on dags from $O(nm)$ to $O(m \log_{2+m/n} n + n \log^3 n)$ time. The proof consists in a reduction from the 2-VDPP on dags to the 2-VDPP on undirected graphs extending some ideas given in the proof of correctness of Lucchesi's and Giglio's linear time reduction from the decision version of the 2-VDPP on dags to the decision version of the 2-VDPP on undirected graphs. Applying the reduction from the 2-EDPP to the 2-VDPP given in [16] results in an $O(m \log_{2+m/n} n + n \log^3 n)$ time algorithm for solving the 2-EDPP on dags.

## 2 Preliminaries

Paths referred to in this paper are always simple paths, i.e. paths on which no vertex appears more often than once. If a vertex $v$ or an edge $e$ is visited by a path $p$, we write $v \in p$ or $e \in p$. For a path $p$ and vertices $a, b \in p$, we let $p[a, b]$ be the sub-path of $p$ from $a$ to $b$. $p(a, b], p[a, b)$, and $p(a, b)$ will denote the sub-paths of $p[a, b]$ starting in the vertex visited immediately after $a$ or, ending in the vertex visited immediately before $b$, or both, respectively. The *length* of a path $p$ is the number of edges visited by $p$ and denoted by $|p|$. Finally, for two paths $p_1$ and $p_2$, $p_1 \circ p_2$ is the concatenation of the two paths.

As for paths, given a tree $T = (V, E)$ and a vertex $v$ or an edge $e$, we write $v \in T$ or $e \in T$ if $v \in V$ or $e \in E$, respectively. The father of a vertex $v \in T$ is denoted by $f_T(v)$.

A *topological numbering* of the vertices of a dag $G = (V, E)$ is an injective mapping from $V$ to $\{1, \ldots, n\}$ such that for each pair $(v, w)$ of vertices for which there is a path from $v$ to $w$, $\tau(v) < \tau(w)$ holds. It is well known, that for each dag $G$ a topological numbering can be computed in linear time (cf. [1]).

## 3 Finding Disjoint Paths Between Pairs of Vertices

Suurballe and Tarjan presented in [14] a data structure which, given a directed graph $G = (V, E)$ and a fixed vertex $s \in V$, for each vertex $v$, can test the existence of two disjoint paths from $s$ to $v$ in constant time. This data structure can be constructed in $O(n + m \log_{2+m/n} n)$ time. It consists of a shortest-path tree $T$ with source node $s$ and stores with each vertex $v \in V$ two vertices $p(v)$ and $q(v)$ which on dags have the following properties:

1. If $\tau$ is a topological numbering of the vertices of $V$, then, for each $v \in V$ with two disjoint paths from $s$ to $v$, $\tau(q(v)) < \tau(v)$ and $(p(v), v) \in E$. 2. If there are two disjoint paths from $s$ to $v$, then there are also two disjoint paths from $s$ to $q(v)$. 3. Two disjoint paths $p_1$ and $p_2$ from $s$ to $v$, if they exist, can be constructed in $O(|p_1| + |p_2|)$ time as follows:

First, mark $v$ and with each marked vertex $x$ also mark $q(x)$. Now, $p_1$ can be constructed in reverse direction starting in $v$ and, when reaching a vertex $x$, following edge $(p(x), x)$ in reverse direction if $x$ is marked, or, if it is not, following edge $(f_T(x), x)$ in reverse direction. $p_2$ can be constructed in the same way, after having un-marked all marked vertices visited by $p_1$.

Let $G' = (V', E')$ be the graph obtained from a dag $G$ by replacing each vertex $v \in V$ with two vertices $v_1$ and $v_2$ and each edge $(u, v)$ with an edge $(u_2, v_1)$ and by adding new edges $(v_1, v_2)$ for every $v \in V$. Then, there are two internally vertex-disjoint paths from a vertex $u \in V$ to a vertex $v \in V$ in $G$, if, and only if, there are two edge-disjoint paths from $u_2$ to $v_1$

in $G'$. Hence, the data structure of Suurballe and Tarjan can be also used to test the existence of two vertex-disjoint paths of a dag $G = (V, E)$ and to construct such paths within the same time bounds as for the edge-disjoint version of the data structure.

In this section we want to show:

**Lemma 1.** *Let $G = (V, E)$ be a dag and $s_1, s_2 \in V$ be fixed. Then, given $k$ pairs of vertices $(u_1, v_1), \ldots, (u_k, v_k)$ it is possible to construct in $O(n \log^2 n + (m + k) \log_{2+(m+k)/(n+k)} n)$ time a data structure that can output, for each pair $(u_l, v_l)$ with $1 \leq l \leq k$ a tuple $(s_1, t_1, s_2, t_2)$ with $\{t_1, t_2\} = \{u_l, v_l\}$ in constant time such that there are two disjoint paths $p_1$, from $s_1$ to $t_1$, and $p_2$, from $s_2$ to $t_2$, if such paths exist. Moreover, the paths themselves can be output in $O(|p_1| + |p_2|)$ time.*

The lemma holds for both, either vertex-disjoint or edge-disjoint paths.

*Proof.* Let $G' = (V', E')$ be the graph obtained from $G$ by adding $k + 1$ vertices $s, w_1, \ldots, w_k$ and $2k + 2$ edges $(s, s_1), (s, s_2), (u_1, w_1), (v_1, w_1), \ldots, (u_k, w_k), (v_k, w_k)$ to $G$. Then our problem can be reduced to the problem of determining, for each $w \in V'$ with two disjoint paths from $s$ to $w$ a tuple $(s_1, y_1, s_2, y_2)$ such that there are two disjoint paths $p_1$ and $p_2$ from $s$ to $w$ with $p_i$ $(i \in \{1, 2\})$ using $(s, s_i)$ as first and $(y_i, w)$ as last edge. From now on we only consider the case, where disjoint means edge-disjoint. The result for the other case follows by the reduction given prior to this lemma.

We start with constructing in $O(n + (m + k) \log_{2+(m+k)/(n+k)} n)$ time the data structure of Suurballe and Tarjan for graph $G'$ with $s$ as fixed source node and define $T, p$, and $q$ to be the shortest-path tree and the mappings described at the beginning of this section. Moreover, we determine in $O(n)$ time a tree $T'$ consisting of all vertices $v \in V'$ for which there are two disjoint paths from $s$ to $v$, $s$ being the root of $T'$, and $f_{T'}(v) = q(v)$ for all $v \in T'$.

In the following, for each $v \in T'$, let $p_1(v)$ and $p_2(v)$ be the two disjoint paths from $s$ to $v$ which would be constructed by Suurballe's and Tarjan's data structure, or more precisely, $p_1(v)$ should be the path visiting $(p(v), v)$ as last edge, and $p_2(v)$ should be the path visiting $(f_T(v), v)$ as last edge. Moreover, for $i \in \{1, 2\}$, we define $r_i(v)$ to be first vertex visited after $s$ on $p_i(v)$.

We now try to determine the vertices $r_1(v)$ for all $v \in V$ (we then also know the vertices $r_2(v)$ for all $v \in V$, since $\{r_1(v), r_2(v)\} = \{s_1, s_2\}$). Therefore, we start a depths-first search in $T'$ and during this depths-first search, when visiting a vertex $y$, we will color the vertices of $T$ such that all vertices $x \neq s$ on the tree path from $s$ to $y$ in $T'$ are colored black if $p_1(x)$ starts with edge $(s, s_1)$, whereas, if $p_1(x)$ starts with edge $(s, s_2)$, vertex $x$ is colored red. All other vertices of $T$ should be colored white. In other words, for a black colored vertex $x$ we have $r_1(x) = s_1$, whereas for a red colored vertex $x$ we have $r_1(x) = s_2$. Note that the red or black colored vertices are exactly the vertices to be marked before the construction of $p_1(y)$ and $p_2(y)$ with the data structure of Suurballe and Tarjan.

When our depths-first search reaches a child $y$ of $s$ in $T'$ it is easy to see that $r_1(y)$ is equal to $y$ if $p(y) = s$, and equal to the first vertex $\neq s$ on the tree path from $s$ to $p(y)$ in $T$, if $p(y) \neq s$. Hence, we know how to color $y$ correctly.

When reaching a vertex $y$ not equal to a child of $s$ in $T'$, we will determine the last red or black colored vertex $x$ before $p(y)$ on the tree path from $s$ to $p(y)$ in $T$. If $x$ not exists, $y$ should be colored black if $(s, s_1)$ is the first edge on the path from $s$ to $y$ in $T$, and, if $(s, s_2)$ is the first edge on this path, $y$ should be colored red. If $x$ exists, from the properties of the data structure of Suurballe and Tarjan given at the beginning of this section it follows that

$p_1(y)[s,y] = p_1(x)[s,x] \circ T[x,p(y)] \circ (p(y),y)$, where $T[x,p(y)]$ denotes the tree path from $x$ to $p(y)$ in $T$. Hence, if by induction we have already shown that all ancestors of $y$ in $T'$ are colored correctly, then $y$ is also colored correctly by coloring it with the same color as $x$.

For an efficient implementation, in a preprocessing step taking $O(m)$ time we determine all $v \in T$ the first vertex $\neq s$ on the tree path from $s$ to $v$ in $T$.

For an efficient computation of the last colored vertex $x$ on tree path from $s$ to a vertex $y$ in $T$, we maintain two copies $T_1$ and $T_2$ of our shortest-path tree $T$. We delete all black and red colored vertices from $T_1$, as well as all black colored vertices from $T_2$. Let $y'$ be the vertex that appears in the middle of the tree path from $s$ to $y$ (note that with an appropriate encoding of the vertices of $T$, $y'$ can be computed in constant time). We then ask whether $y$ is reachable from $y'$ in $T_1$. If this is the case, $x$ does not exist or lie on the tree path from $s$ to $y'$ in $T$. Otherwise, our search can be reduced to the tree path from $y'$ to $y$ in $T$. In other words, $x$ can be determined by a binary search. We can also identify the color of $x$ by testing whether $y$ is reachable from $f_T(x)$ in $T_2$. We use the dynamic data structure of Holm, de Lichtenberg, and Tarjan[5] for updating the trees $T_1$ and $T_2$ and for answering our connectivity queries. This data structure allows us to delete a vertex with $r$ adjacent edges or to reinsert such a vertex in $O(r \log^2 n)$ amortized time and to decide whether two vertices are connected in $O(\log n / \log\log n)$ worst case time.

Since the number of edge deletions and reinsertions needed for the computation of the vertices $r_1(v)$ for all $v \in V$ is bounded by $O(n)$ and we only have to answer $O(n \log n)$ queries, the construction time for our data structure can be bounded by $O(n \log^2 n + (m + k) \log_{2+(m+k)/(n+k)} n)$ time. Given a vertex $v$, the paths $p_1(v)$ and $p_2(v)$ can be output with the data structure of Suurballe and Tarjan in $O(|p_1| + |p_2|)$ time. □

## 4 Solving the 2-VDPP on dags

In this section we present an $O(m \log_{2+m/n} n + n \log^3 n)$-time algorithm for solving the 2-VDPP on a dag $G = (V, E)$. We extend some ideas already used by Lucchesi and Giglio in [6] to prove the correctness of their linear time reduction from the decision version of the 2-VDPP on dags to the decision version of the 2-VDPP on undirected graphs.

As observed by Thomassen [18], it is easy to see that, given an instance $I = (G, s_1, s_2, t_1, t_2)$ of the 2-VDPP on dags, the following reductions will not change the solvability of the 2-VDPP (i.e. there are two disjoint paths $p_1$, from $s_1$ to $t_1$, and $p_2$, from $s_2$ to $t_2$, before the reduction, iff the same is true after the reduction):

- Delete an edge $e \in \{(v, s_i) \mid v \in V, 1 \leq i \leq 2\} \cup \{(t_i, v) \mid v \in V, 1 \leq i \leq 2\}$.
- Delete a vertex $v \notin \{s_1, s_2, t_1, t_2\}$ with its adjacent edges that has no adjacent edges entering $v$ or no adjacent edges leaving $v$.
- If there is a vertex $v \in V - \{s_1, s_2\}$ with only one edge $(v, w)$ leaving $v$, delete $v$ from $G$ and replace each edge $(u, v)$ with a new edge $(u, w)$ (if edge $(u, w)$ already exists, just delete edge $(u, v)$ from $G$).
- If there is a vertex $v \in V - \{t_1, t_2\}$ with only one edge $(u, v)$ entering $v$, delete $v$ from $G$ and replace each edge $(v, w)$ with a new edge $(u, w)$ (if edge $(u, w)$ already exists, just delete edge $(v, w)$ from $G$).

Let us call an instance $I$ of the 2-VDPP to be *irreducible* if none of the reductions above are applicable to G. Then it is easy to prove the following lemma:

**Lemma 2.** *For each instance $I_1 = (G, s_1, s_2, t_1, t_2)$ of the 2-VDPP with $G = (V, E)$ being a dag one can construct in $O(m)$ time an irreducible instance $I_2 = (G', s_1, s_2, t_1, t_2)$ of the 2-VDPP on dags with $G' = (V', E')$ such that,*

- *$I_1$ has a solution iff $I_2$ has a solution,*
- *$|V'| = O(|V|)$ and $|E'| = O(|E|)$,*
- *given a solution to $I_2$ we can solve $I_1$ in $O(|V| + |E|)$ time.*

According to Lemma 2, we will from now on concentrate on solving the 2-VDPP on irreducible instances and describe an algorithm for solving such instances. This algorithm will make use of the following technical lemmas:

**Lemma 3.** *Let $G = (V, E)$ be a dag in which each vertex has an out-degree of at least two with the exception of two vertices $x$ and $y$ which have an out-degree of 0 and are reachable from every vertex $v \in V - \{x, y\}$. Then, for each pair $v, w$ of vertices there are two vertex-disjoint paths $p_1$ and $p_2$ with one path starting in $v$ and the other one in $w$ and with one path ending in $x$ and the other one in $y$. In particular this holds for all dags $G$ with $(G, s_1, s_2, t_1, t_2)$ being an irreducible instance of the 2-VDPP, if we let $x := t_1$ and $y := t_2$.*

*Proof.* If either $v$ or $w$ is equal to either $x$ or $y$, say w.l.o.g. $v = x$, we only need to search for a path from $w$ to $y$ not visiting vertex $x$. Since every vertex $u \in V - \{x, y\}$ has an out-degree of at least two, such a path must exist. Hence, let us assume that $w, v \notin \{x, y\}$.

Let $\tau$ be a topological numbering of the vertices of $G$. It is clear that $x$ and $y$ must have the largest topological numbers, namely, $n - 1$ and $n$. Given $v$ and $w$, we construct step by step two disjoint paths from $v$ and $w$ to $x$ and $y$ starting with two paths $p_v$ from $v$ to $v$ and $p_w$ from $w$ to $w$ of zero length. We always try to maintain the following invariant: If $r$ and $u$ are the endpoints of $p_v$ and $p_w$, where $\tau(r) < \tau(u)$ and $r \notin \{x, y\}$, then the path of $p_v$ and $p_w$ with endpoint $u$ visits no vertex $z$ with $\tau(z) > \tau(r)$ before $u$. It is obvious that this invariant holds immediately after the initialization of $p_v$ and $p_w$. Defining $r$ and $u$ as above we can add an edge $(r, z)$ to the path ending in $r$ such that the paths remain vertex-disjoint, since $r$ has an out-degree of at least two. Now, either the invariant holds after adding edge $(r, z)$, and this means we can add an further edge to one of the paths, or $\{r, u\} = \{x, y\}$ and, hence, $p_v$ and $p_w$ are two disjoint paths from $v$ and $w$ to $x$ and $y$. □

**Lemma 4. (Thomassen [18])** *If $G$ is a dag with $I = (G, s_1, s_2, t_1, t_2)$ being an irreducible instance of the 2-VDPP, then, for each vertex $v \in V - \{s_1, s_2, t_1, t_2\}$, there exist four paths $p_1$ from $s_1$ to $v$, $p_2$ from $s_2$ to $v$, $p_3$ from $v$ to $t_1$, and $p_4$ from $v$ to $t_2$ such that the only vertex visited by more than one of the four paths is $v$.*

*Proof.* Let us consider two outgoing edges $(v, u)$ and $(v, w)$ from $v$. From Lemma 3 we know that there are two disjoint paths from the vertices $u$ and $w$ to the vertices $t_1$ and $t_2$. Adding the edges $(v, u)$ and $(v, w)$ to these paths results in two disjoint paths leading from $v$ to the vertices $t_1$ and $t_2$. In order to construct disjoint paths from $s_1$ and $s_2$ to $v$ we apply Lemma 3 to the graph obtained from $G$ by replacing each edge $(v, w)$ by its reverse edge $(w, v)$. □

In the following, we let $G = (V, E)$ be a dag and $I = (G, s_1, s_2, t_1, t_2)$ be an irreducible instance of the 2-VDPP. Moreover, we define $U(G)$ to be the undirected graph obtained from $G$ by replacing each directed edge $(u, v)$ of $G$ with an undirected edge $\{u, v\}$. In the first step of our algorithm, we determine two simple disjoint paths $p_1$, from $s_1$ to $t_1$, and $p_2$, from $s_2$ to $t_2$, in $U(G)$.

**Table 1.** The path-replacement in the different sub-cases

| Sub-case | Description: For $i,j$ with $\{1,2\}=\{i,j\}$ | Replacements |
|---|---|---|
| 1a | $v \in p_i, v' \in r_j$ | $p_i^* := p_i[s_i,v] \circ r_i[v,t_i]$ |
| 2a | $v \in p_i, v' \in r_j$ | $p_j^* := p_j[s_j,v'] \circ r_j[v',t_j]$ |
| 1b.$\alpha$ | $u \in p_i[c_i,t_i], u' \in q_j$ | $p_i^* := q_i[s_i,u] \circ p_i[u,t_i]$ |
| 1b.$\beta$ | $u \in p_i[s_i,c_i), u' \in q_j$ | $p_j := q_j[s_j,u'] \circ p_j[u',t_j]$ |
| 2b | $u \in p_i, u' \in q_j$ | |
| 1c.$\alpha$ | $u,v \in p_i, u' \in q_i, v' \in r_i, u \notin p_i(d_i,ti]$ | $p_i^* := q_i[s_i,u'] \circ p_j[u',v'] \circ r_i[v',t_i]$ |
| 1c.$\beta$ | $u,v \in p_i, u' \in q_i, v' \in r_i, u \in p_i(d_i,ti]$ | $p_j^* := q_j[s_j,u] \circ p_i[u,v] \circ r_j[v,t_j]$ |
| 2c | $u,v \in p_i, u' \in q_i, v' \in r_i$ | |
| 1d | $u \in p_i, v \in p_j, u' \in q_i,\ v' \in r_j$ | $p_i^* := q_i[s_i,u'] \circ p_j[u',v] \circ r_i[v,t_i]$ |
| 2d | $u \in p_i, v \in p_j, u' \in q_i,\ v' \in r_j$ | $p_j^* := q_j[s_j,u] \circ p_i[u,v'] \circ r_j[v',t_j]$ |

Like Lucchesi and Giglio we call an edge $(u,v)$ visited in this direction by $p_1$ or $p_2$, a *forward edge*, if $(u,v) \in E$, and, if not (i.e. $(v,u) \in E$), a *reverse edge*. For two consecutive edges $(u,v)$ and $(v,w)$ on $p_1$ or $p_2$, one being a forward and the other one being a reverse edge, $v$ will be reffered to as a *switch*.

Lucchesi and Giglio in [6] proved that there is a choice of four vertices $u, u', v$, and $v'$ with $u$ and $v$ being switches which makes it possible to replace $p_1$ and $p_2$ with two new paths $p_1^*$ and $p_2^*$ such that all switches of $p_1^*$ and $p_2^*$ are also switches of $p_1$ or $p_2$ and such that $u$ or $v$ is no longer a switch on $p_1^*$ or $p_2^*$. The main idea of our algorithm consists of dividing our algorithm into several rounds and of choosing in each round the vertices $u, u', v$, and $v'$ much more carefully so that the path replacements of $p_1$ and $p_2$ removes at least a constant fraction of all switches.

Let us sketch what is done in each round. For $i \in \{1,2\}$, let $n_i$ be the number of switches on $p_i$ at the beginning of the round, let $c_i$ be the vertex on $p_i$ visited immediately after the $\lfloor \frac{1}{4} n_i \rfloor$-th switch of $p_i$ and let $d_i$ be the vertex on $p_i$ visited immediately before $(n_i - \lfloor \frac{1}{4} n_i \rfloor)$-th switch of $p_i$. If $\lfloor \frac{1}{4} n \rfloor = 0$, we let $c_i := s_i$ and $d_i := t_i$.

We now choose four vertices $u, v, u', v'$ as follows: Let $\tau$ be a topological numbering of the vertices of $G$. Like Lucchesi and Giglio in [6] we define $v$ to be the switch with largest topological number among all switches on $p_1$ and $p_2$, but unlike Lucchesi and Giglio we let $v'$ be first vertex $x$ with $\tau(x) > \tau(v)$ visited by the path $p_1$ or $p_2$ which does not visit $v$. We then distinguish between two main cases: Case 1, where $v \in p_1[s_1,c_1)$ or $v \in p_2[s_2,c_2)$, and Case 2, where $v \in p_1[c_1,t_1]$ or $v \in p_2[c_2,t_2]$. In Case 1, like Lucchesi and Giglio, we define $u$ to be the switch with the lowest topological number on $p_1$ or $p_2$, whereas in Case 2, unlike Lucchesi and Giglio, we let $u$ be the switch on $p_1[c_1,t_1]$ or $p_2[c_2,t_2]$ having the smallest topological number among all switches on these sub-paths. In both cases we let $u'$ be the last vertex $x$ with $\tau(x) < \tau(u)$ visited by the path $p_1$ or $p_2$ not visiting $u$. Moreover, again in both cases, we define $q_1$ and $q_2$ to be two disjoint paths from $s_1$ and $s_2$ to $u$ and $u'$ such that $q_1$ starts in $s_1$ and, hence, $q_2$ starts in $s_2$, and, similarly, let $r_1$ and $r_2$ be two disjoint paths from $v$ and $v'$ to $t_1$ and $t_2$ such that $r_1$ ends in $t_1$ and $r_2$ ends in $t_2$. These paths must exist because of Lemma 3.

Depending on the positions of the vertices $u, u', v, v'$ on the paths $p_1$ and $p_2$ we consider different sub-cases and replace $p_1$ and $p_2$ with two paths $p_1^*$ and $p_2^*$ as shown in Table 1. For $i \in \{1,2\}$, sub-cases with prefix number $i$ should be a sub-case of Case $i$. The new paths are disjoint:

**Lemma 5.** *$p_1^*$ and $p_2^*$ are disjoint.*

*Proof.* For the Cases 1a, 2a, 1b.$\alpha$, 1b.$\beta$, and 2.b this follows from the fact that the sub-paths of $p_1$ and $p_2$ used for the construction of $p_1^*$ and $p_2^*$ apart from $u'$ and $v'$ visit only vertices $x$ with $\tau(x) \leq \tau(v)$ (Cases 1a, 2a) or only vertices $x$ with $\tau(x) \geq \tau(u)$ (Cases 1b.$\alpha$, 1b.$\beta$, 2b). Let $p_1'$ and $p_2'$ be the sub-paths of $p_1$ and $p_2$, respectively, that were used for the construction of $p_1^*$ and $p_2^*$ in one of the remaining cases. Then the disjointness from $p_1^*$ and $p_2^*$ in the remaining cases follows if we can show that $\tau(u) \leq \tau(x) \leq \tau(v)$ holds for all $x \in p_1'$ and all $x \in p_2'$ with $x \notin \{u', v'\}$. It is easy to see that this holds, if, for every ordered pair of vertices $(x, y) \in \{(u, v'), (u', v), (u', v')\}$ with $x, y \in p_i'$ for an $i \in \{1, 2\}$, $x$ appears before $y$ on $p_i'$. But this last statement is true since $v'$ must appear after the last switch on $p_1$ or $p_2$, whereas $u'$, in Case 1, must appear before the first switch on $p_1$ or $p_2$, and, in Case 2, must appear before the first switch on $p_1[c_1, t_1]$ or $p_2[c_2, t_2]$, and, therefore, before $v$ or $v'$ on $p_1$ or $p_2$. □

From the definition of $u$ and $v$ as vertices with the smallest or largest topological number it follows that after the path replacements shown in Table 1 $u$ or $v$ can no longer be a switch of $p_1$ or $p_2$. Unfortunately, we can not guarantee that much more switches are deleted from $p_1$ and $p_2$ in the Cases 1b.$\beta$, 1c.$\beta$, or 2a. Therefore, in these cases the idea is to consider not only one round but a number $k$ of rounds such that in the first $k-1$ rounds we are in one of the Cases 1b.$\beta$, 1c.$\beta$, or 2a, and in the last round we are in one of the other cases.

We will from now on consider the $k$ rounds as exactly one round sometimes also called *super-round* and the $k$ rounds as *sub-rounds* of this round. For a simpler implementation of the sub-rounds we will not update the vertices $c_i$ and $c_j$, after each of the first $k-1$ sub-rounds. There is one exception: In a sub-round corresponding to Case 1c.$\beta$ we replace $c_i$ with $d_i$ and $d_i$ with $c_i$ (since $p_i$ after the replacement visits the vertices between $c_i$ and $d_i$ in reverse direction). The $k$-th sub-round then guarantees that enough switches are being removed from $p_1$ and $p_2$ in each super-round.

More precisely, from the replacements given in Table 1 we can conclude that after each round - i.e. after each super-round in the Cases 1b.$\beta$, 1c.$\beta$, or 2a - at least $1 + \min\{\lfloor \frac{1}{4} n_1 \rfloor, \lfloor \frac{1}{4} n_2 \rfloor\}$ switches (or $1 + \max\{\lfloor \frac{1}{4} n_1 \rfloor, \lfloor \frac{1}{4} n_2 \rfloor\}$ switches if $n_1 = 0$ or $n_2 = 0$) are removed from $p_1$ and $p_2$. For example, in Case 1c.$\alpha$ at least all switches of $p_i(d_i, t_i]$ are removed from $p_i$. Thus our algorithm terminates after $O(\log n)$ rounds with two disjoint paths $p_1$, from $s_1$ to $t_1$, and $p_2$, from $s_2$ to $t_2$. We now show that each round can be implemented efficiently:

**Lemma 6.** *Each round has a running time of* $O(n \log^2 n + m \log_{2+m/n} n)$.

*Proof.* It is obvious that for each round (/super-round) the numbers $n_1$ and $n_2$ as well as the vertices $c_i, d_i, c_j$, and $d_j$ and therefore the boundary vertices $u, u', v, v'$ (of the first sub-round in the case of a super-round) can be computed in $O(n)$ time. With standard network flow techniques two disjoint paths from the vertices $s_1$ and $s_2$ to the vertices $u$ and $u'$ as well as two disjoint paths leading from the vertices $v$ and $v'$ to the vertices $t_1$ and $t_2$ can be computed in $O(m)$ time. Given these paths, it is easy to determine in which case we are and to implement the path replacements for the Cases 1a, 1b.$\alpha$, 1c.$\alpha$, 1d, 2b, 2c, and 2.d, again in linear time.

We now consider the time complexity of the Cases 1b.$\beta$, 1c.$\beta$, and 2a. In the following, when talking about the paths $p_1$ and $p_2$ or the boundary vertices, if we mean the paths at the beginning of the $l$-th sub-round or the boundary vertices in the $l$-th sub-round we denote them by $p_1^l, p_2^l, u^l, u'^l, v^l$, or $v'^l$, respectively, and if we mean the paths after the last sub-round we write $p_1^{k+1}$ and $p_2^{k+1}$.

Let us define the *original part* of $p_1^l$ and $p_2^l$ to be the part of $p_1^l$ and $p_2^l$ that is equal to the corresponding part of $p_1^1$ or $p_2^1$. More precisely, if before the first sub-round we mark all

vertices of $p_1^1$ and $p_2^1$ and in the $j$-th sub-round when replacing $p_1^j$ and $p_2^j$ with $p_1^{j+1}$ and $p_2^{j+1}$ we un-mark all edges not lying on the sub-paths of $p_1^j$ and $p_2^j$ used for the construction of $p_1^{j+1}$ and $p_2^{j+1}$, then the original part of $p_1^l$ ($p_2^l$) is the sub-path of $p_1^l$ ($p_2^l$) consisting of the marked edges.

We next want to show that the boundary vertices of each sub-round must lie on the original parts of this sub-round, which will be useful for an efficient implementation of the sub-rounds. It is obvious that, for $l \in \{1, \ldots, k\}$, $u^l$ and $v^l$ must lie on the original part of $p_1^l$ and $p_2^l$ since all switches of $p_1^l$ and $p_2^l$ lie on the original part of these paths.

For an $l \in \mathbb{N}$, let us define numbers $l_1$ and $l_2$ such that the $l_1$-th sub-round is the last sub-round before the $l$-th sub-round in which we are in Case 1b.$\beta$ or 1c.$\beta$ and the $l_2$-th sub-round is the last sub-round before the $l$-th sub-round in which we are in Case 2a or 1c.$\beta$ ($l_1$ or $l_2$ should be 0 if no such sub-round exists). Then by induction one can show that the endpoints of the original parts of $p_1^l$ and $p_2^l$ consist of the vertices $u^{l_1}, u'^{l_1}, v^{l_2}$, and $v'^{l_2}$, where we define $u^0 = s_1, u'^0 = s_2, v^0 = t_1$, and $v'^0 = t_2$. Moreover, again by induction one can show that $\tau(u^{l_1}) \leq \tau(x) \leq \tau(v^{l_2})$ holds for all vertices $x \notin \{u'^{l_1}, v'^{l_2}\}$ on the original parts of $p_1^l$ and $p_2^l$. Now, from $\tau(u'^{l_1}) < \tau(u^{l_1}) \leq \tau(u^l) \leq \tau(v^l) \leq \tau(v^{l_2}) \leq \tau(v'^{l_2})$ we can conclude that the vertices $u'^l$ and $v'^l$ must appear after a vertex $x \in \{u^{l_1}, u'^{l_1}\}$ on $p_1^l$ or $p_2^l$ or be equal to $x$ and they must appear before a vertex $y \in \{v^{l_2}, v'^{l_2}\}$ on $p_1^l$ or $p_2^l$ or be equal to $y$. Therefore, $u'^l$ and $v'^l$ lie on the original part of $p_1^l$ or $p_2^l$. We can use the knowledge that the boundary vertices lie always on the original part of $p_1$ or $p_2$ and that this part is always a sub-paths of $p_1^1$ or $p_2^1$ for an efficient computation of the boundary vertices:

Knowing the original parts of $p_1$ and $p_2$ for each sub-round, we can easily compute $v^l$ for all $l \in \{1, \ldots, k\}$ if, before starting the first sub-round, we construct in $O(n)$ time a list of all switches on $p_1$ and $p_2$ sorted by their topological numbers. We then only need to delete repeatedly the vertex with the largest topological number from this list until we find a vertex with $x$ lying on the original part of $p_1$ or $p_2$. We can always start the search with the last vertex deleted in the previous sub-round. Hence, the time needed to compute the boundary vertex $v$ taken over all sub-rounds is bounded by $O(n)$, and, in nearly the same way, this also holds for the boundary vertex $u$.

We still have to explain how we can compute $u'^l$ and $v'^l$ for all $l \in \{1, \ldots, k\}$: For $i \in \{1, 2\}$, let $w_i$ be the first switch on $p_i^l$ and let $z_i$ be the first switch on $p_i^l[c_i, t_i]$, or, if there is no switch on $p_i^l$, let $w_i = z_i = t$. Prior to the first sub-round we compute, for all $i \in \{1, 2\}$ and all switches $x \in p_i^1$, two vertices $l(x)$ and $l'(x)$ such that $l(x)$ is the last visited vertex $y$ with $\tau(y) < \tau(x)$ on $p_j^1[s_j, w_j]$ and $l'(x)$ is the last visited vertex $y$ with $\tau(y) < \tau(x)$ on $p_j^1[s_j, z_j]$. From the fact that $u'^l$ lies on the original part of $p_1^l$ or $p_2^l$, and from the fact that $u'^l$ in Case 1 must appear before $w_1$ or $w_2$ and in Case 2 before $z_1$ or $z_2$ on one of the paths $p_1^l$ or $p_2^l$, we can conclude that $u'^l = l(x)$ in Case 1, and $u'^l = l'(x)$ in Case 2. $l(x)$ and $l'(x)$ can be computed efficiently:

**Lemma 7.** *Computing for all switches $x \in p_1^1$ and $x \in p_2^1$ the values $l(x)$ and $l'(x)$ takes $O(n)$ time.*

*Proof.* Computing $l(x)$, for all switches of $p_i$ ($i \in \{1, 2\}$), can be done in $O(n)$ time if we consider the switches in topological decreasing order: For determining $l(x)$ for the switch $x$ of $p_i$ with the largest topological number, we step through the vertices of $p_j$ in reverse direction from $w_j$ to $s_j$ until we find a first vertex $y$ with $\tau(y) < \tau(x)$. $y$ must then be equal to $l(x)$, and, for all vertices $z \in p_j(l(x), w_j]$, we have $\tau(z) > \tau(y)$. Hence, if we search for the vertex $l(x')$ with $x'$ being the switch with the second largest topological number on $p_i$, we continue the walk along the path $p_j$ now starting in $l(x)$. We can conclude that computing the values $l(x)$

for all switches of $p_1^1$ or $p_2^1$ can be done in $O(n)$ time. Similarly, the values $l'(x)$ for all switches can be computed in $O(n)$ time. □

$v'^l$ can be computed in the same efficient way. We just have seen, that if we know for each sub-round in which Case we are, i.e. if we know the original parts of $p_1^l$ and $p_2^l$, we can efficiently compute the vertices $u^l, u'^l, v^l$, and $v'^l$. To decide, for each sub-round, in which case we are, the super-round is split into two phases. In the first phase, if in a sub-round $u$ and $v$ lie on $p_1$ and $p_2$ in such a way that we might be in Case 1b.$\beta$, 1c.$\beta$, or 2a, we always assume that we are in this case and, under this assumption, we compute the boundary vertices for the next sub-round. For example, if $v \in p_i[s_i, c_i)$ and $u \in p_j[s_j, c_j)$ with $\{i, j\} = \{1, 2\}$ we assume that we are in Case 1b.$\beta$ (note that we will never encounter more than one of the three cases).

After the first phase we construct the data structure described in Lemma 1 with $(u_1, v_1), \ldots, (u_k, v_k)$ being equal to the pairs of boundary vertices $(u, u')$ of each sub-round considered in the first phase of our super-round.

In the second phase, starting again with the first sub-round we use this data structure to determine for each pair $(u^l, u'^l)$ of boundary vertices a tuple $(s_1, w_1, s_2, w_2)$ with $\{w_1, w_2\} = \{u^l, u'^l\}$ such that there are two disjoint paths $r_1$, from $s_1$ to $w_1$, and $r_2$, from $s_1$ to $w_2$, and in the same way we can construct a tuple $(w_1, t_1, w_2, t_2)$ with $\{w_1, w_2\} = \{v^l, v'^l\}$ such that there are two disjoint paths from $w_1$ to $t_1$ and $w_2$ to $t_2$. We finally test whether we are in one of the Cases 1b.$\beta$, 1c.$\beta$, or 2a and, therefore, have correctly computed the boundary vertices of the next sub-round. If we are in one of the other cases we stop the computation of boundary vertices since then we must be in the last sub-round of the super-round.

We finally want to determine the paths $p_1^{k+1}$ and $p_2^{k+1}$ resulting from the replacement of $p_1$ and $p_2$ in the last sub-round of our super-round. If in the last sub-round we are in one of the cases 1c.$\alpha$, 2.c, 1.d, or 2d, $p_1$ and $p_2$ are replaced by three pairs of disjoint paths: $r_1$ and $r_2$, from $s_1$ and $s_2$ to $u^k$ and $u'^k$, $q_1$ and $q_2$, from $v^k$ and $v'^k$ to $t_1$ and $t_2$, and, two sub-paths of the original parts of $p_1^k$ and $p_2^k$. We can read off these paths from the data structure of Lemma 1 and from the paths $p_1^1$ and $p_2^1$ in $O(n)$ time. Even, if in last sub-round we are in one of the Cases 1a, 1b.$\alpha$, or 2b, we can construct $p_1^{k+1}$ and $p_2^{k+1}$ in $O(n)$ time. For details see the appendix of this paper. □

**Theorem 8.** *On dags the 2-VDPP is solvable in $O(m \log_{2+m/n} n + n \log^3 n)$ time.*

*Proof.* As shown, our algorithm for solving the 2-VDPP consists of $O(\log n)$ rounds with a running time of $O(n \log^2 n + m(\log_{2+m/n} n))$. This leads to a running time of $O(n \log^3 n + m(\log n)(\log_{2+m/n} n)$. The stated time bound follows, if the 2-VDPP on dags can be reduced to the 2-VDPP on dags with only $O(n)$ edges.

In [6] Lucchesi and Giglio have shown that two disjoint paths $p_1$, from $s_1$ to $t_1$, and $p_2$, from $s_2$ to $t_2$, on a dag $G = (V, E)$ can be constructed from two disjoint paths in $U(G)$ by replacing sub-paths of $p_1$ and $p_2$ by sub-paths of a set $S$ of disjoint paths. More precisely, if we add extra vertices $x$ and $y$ as well as four extra edges $(x, s_1), (x, s_2), (t_1, y)$, and $(t_2, y)$ to $G$, $S$ can be chosen arbitrarily as long as $S$ consists of two disjoint paths from $x$ to $v$ as well as two disjoint paths from $v$ to $y$ for every $v \in V$. Such paths must exist because of Lemma 4.

As disjoint paths from $x$ to the vertices $v \in V$ let us choose the paths that would be constructed by the data structure of Suurballe and Tarjan given in [14]. These paths can only consist of the edges of the shortest-path tree $T$ and of edges of the form $(p(w), w)$ with $T$ and $p$ being defined as described in the beginning of Section 3. Consequently, the graph containing these $O(n)$ edges plus $O(n)$ edges needed for the construction of disjoint paths from vertices

$v \in V$ to $y$, and containing the edges of $p_1$ and $p_2$ is a subgraph of $G$ on which the 2-VDPP is solvable, but which consists of only $O(n)$ edges. □

## References


1. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms (2nd edition),* MIT Press, Cambridge, MA, 2001.
2. G. Di Battista, R. Tamassia, and L. Vismara, Output-sensitive reporting of disjoint paths, *Algorithmica* **23** (1999), pp. 302–340.
3. Y. Dinitz and J. Westbrook, Maintaining the classes of 4-edge-connectivity in a graph on-line, *Algorithmica* **20** (1998), pp. 242–276.
4. S. Fortune, J. Hopcroft, and J. Wyllie, The directed subgraph homeomorphism problem, *Theoret. Comput. Sci.* **10** (1980), pp. 111–121.
5. J. Holm, K. de Lichtenberg, and M. Thorup, Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity, *J. ACM* **48** (2001), pp. 723–760.
6. C. L. Lucchesi and M. C. M. T. Giglio, On the irrelevance of edge orientations on the acyclic directed two disjoint paths problem, IC Technical Report DCC-92-03, Universidade Estadual de Campinas, Instituto de Computação, 1992.
7. T. Ohtsuki, The two disjoint path problem and wire routing design, Proc. Symposium on Graph Theory and Applications, Lecture Notes in Computer Science, Vol. 108, Springer, Berlin, 1981, pp. 207–216.
8. L. Perković and B. Reed, An improved algorithm for finding tree decompositions of small width, *International Journal of Foundations of Computer Science (IJFCS)* **11** (2000), pp. 365–372.
9. Y. Perl and Y. Shiloach, Finding two disjoint paths between two pairs of vertices in a graph, *J. ACM* **25** (1978), pp. 1–9.
10. N. Robertson and P. D. Seymour, Graph minors. XIII. The disjoint paths problem, *J. Comb. Theory, Ser. B*, **63** (1995), pp. 65–110.
11. A. Schrijver, A group-theoretical approach to disjoint paths in directed graphs, *CWI Quarterly* **6** (1993), pp. 257–266.
12. P. D. Seymour, Disjoint paths in graphs, *Discrete Math.* **29** (1980), pp. 293–309.
13. Y. Shiloach, A polynomial solution to the undirected two paths problem, *J. ACM* **27** (1980), pp. 445–456.
14. J. W. Suurballe and R. E. Tarjan, A quick method for finding shortest pairs of disjoint paths. *Networks* **14** (1984), pp. 325-336.
15. R. E. Tarjan, and J. van Leeuwen, Worst-case analysis of set union algorithms, *J. ACM* **31** (1984), pp. 245–281.
16. T. Tholey, Solving the 2-disjoint paths problem in nearly linear time. Proc. 21st Annual Symposium on Theoretical Aspects of Computer Science (STACS 2004), Lecture Notes in Computer Science Vol. 2996, Springer-Verlag, Berlin, 2004, pp. 350–361.
17. C. Thomassen, 2-linked graphs, *Europ. J. Combinatorics* **1** (1980), pp. 371–378.
18. C. Thomassen, The 2-linkage problem for acyclic digraphs, *Discrete Math.* **55** (1985), pp. 73–87.


## 5 Appendix

### Computation of the paths $p_1^{k+1}$ and $p_2^{k+1}$, if we are in Case 1a, 1b.$\alpha$, or 2b.

Let us assume that in the last sub-round we are in Case 1b.$\alpha$. The other cases can be handled in a similar way. Since in the last sub-round we are in Case 1b.$\alpha$ the endpoints of the original parts of $p_1^{k+1}$ and $p_2^{k+1}$ consist of the vertices $u^k$ and $u'^k$ as well as of two further vertices $x$ and $y$. Note that $x, y \notin \{u^k, u'^k\}$ ($x \neq u^k \neq y$ since $x$ and $y$ are not switches, $x \neq u'^k \neq y$ since $u'^k$ must appear before $v^k$ and $v'^k$ and, therefore, before $x$ or $y$ on $p_1^k$ or $p_2^k$). For $z \in \{x, y\}$, there must be an $i$-th sub-round having $z$ as one of its boundary vertices and making $z$ to an endpoint of the original part of $p_1^j$ and $p_2^j$ for all $j \in \{i+1, \ldots, k+1\}$. [2] Since $z$ is visited after $u^k$ or $u'^k$ on $p_1^k$ or $p_2^k$, we have $\tau(z) > \tau(u^k) > \tau(u'^k)$ and, therefore,

[2] To keep our proof simple, suppose there is a sub-round 0 of case 1b.$\beta$ having $s_1, s_2, t_1$, and $t_2$ as boundary vertices and making them to the endpoints of the original parts of $p_1^1$ and $p_2^1$.

$z = v^i$ or $z = v'^i$. Moreover, in the $i$-th sub-round, we must be in one of the Cases 1c.$\beta$ or 2a. Choose $l$ maximal such the $l$-th sub-round has $x$ or $y$ as one its boundary vertices $v^l$ or $v'^l$ and the sub-round is of kind 1c.$\beta$ or 2.a. Let us assume for a moment that $x$ is the vertex with $x \in \{v^l, v'^l\}$. We can conclude that $y \in \{v^l, v'^l\}$ must also hold, since, otherwise, for all $j \in \{i+1, \ldots, k+1\}$, either the original part $p_1^{j+1}$ or the original part of $p_2^{j+1}$ either does not contain $y$ or it contains $y$, but does not contain $y$ as an endpoint of an original part (note, there is no further sub-round having $y$ as one of its boundary vertices).

Suppose now that $x \in p_1^{l+1}$ - otherwise rename $x$ and $y$ - and say $u^k \in p_1^{l+1}$ (the case $u'^k \in p_1^{l+1}$ can be handled in the same way).

We show that $p_1^{l+1}[u^k, t_1] = p_1^k[u^k, t_1]$ and $p_2^{l+1}[u'^k, t_2] = p_2^k[u'^k, t_2]$.

$p_1^{l+1}[u^k, x] = p_1^k[u^k, x]$ and $p_2^{l+1}[u'^k, y] = p_2^k[u'^k, y]$ holds, since $p_1^k[u^k, x]$ and $p_2^k[u'^k, y]$ are sub-paths of the original parts of $p_1^k$ and $p_2^k$. Hence, if one of the above equations does not hold, then there is a $j \in \{l+1, k\}$ such that the $j$-th sub-round must have a boundary vertex $b$ on $p_1^j[x, t_1] = p_1^{l+1}[x, t_1]$ or on $p_2^j[y, t_2] = p_2^{l+1}[y, t_2]$ causing one of this sub-paths to change. Since $\tau(b) > \tau(u'^k)$ and $b$ cannot be a switch, we can conclude that $b = v'^j$ and we are in one of the Cases 1c.$\beta$ or 2$a$. It follows that $v^j$ must lie on the part of $p_1^j$ or $p_2^j$ containing switches, i.e. on $p_1^{l+1}[u^k, x]$ and $p_2^{l+1}[u^k, x]$, and as a consequence of the path replacements in the cases Cases 1c.$\beta$ or 2$a$ one of these sub-paths will be changed so that not the complete sub-path $p_1^{l+1}[u^k, x]$ or $p_2^{l+1}[u^k, x]$ belongs to the original part of the following sub-rounds. A contradiction.

Thus, we know that,

$$\begin{aligned} p_1^{k+1} &= q_1[s_1, u^k] \circ p_1^k[u^k, t_1] \\ &= q_1[s_1, u^k] \circ p_1^{l+1}[u^k, t_1] \\ &= q_1[s_1, u^k] \circ p_1^l[u^k, x] \circ r_1[x, t_1] \\ &= q_1[s_1, u^k] \circ p_1^1[u^k, x] \circ r_1[x, t_1] \end{aligned}$$

and

$$\begin{aligned} p_2^{k+1} &= q_2[s_2, u'^k] \circ p_2^k[u'^k, t_1] \\ &= q_2[s_2, u'^k] \circ p_2^{l+1}[u'^k, t_1] \\ &= q_2[s_2, u'^k] \circ p_2^l[u'^k, y] \circ r_2[y, t_2] \\ &= q_2[s_2, u'^k] \circ p_2^1[u'^k, y] \circ r_2[y, t_2], \end{aligned}$$

where $r_1$ and $r_2$ are two disjoint paths from the vertices $x$ and $y$ to $t_1$ and $t_2$, hence, after identifying $l$ in linear time, $p_1^{k+1}$ and $p_2^{k+1}$ can be read off from the data structure of Lemma 1 and the paths $p_1^1$ and $p_2^1$ in linear time.