Finding Disjoint Paths on Directed Acyclic Graphs

Torsten Tholey

23.08.2005

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 a tuple (s_1, s_2, t_1, t_2) of four vertices, 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.

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 a tuple (s_1, s_2, t_1, t_2) of four vertices, 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. [1] and [2]), 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 $t_{\pi(i)}$ such that π is a permutation 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 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 [1] have shown that allowing a preprocessing time of $O(m+n)$ (if $k=2$) or $O(n^2)$ (if $k=3$) one can construct a data structure that can test the existence of k vertex-disjoint paths between each pair of 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 [2].

For a directed graph $G = (V, E)$ and a fixed vertex $s \in V$, Suurballe and Tarjan [13] 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.

Another interesting paths finding problem is the k-disjoint paths problem. In this problem we are given a tuple $(s_1, t_1, \ldots, s_k, t_k)$ of $2k$ vertices 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 on undirected graphs where given by Ohtsuki [6], Seymour [11], Shiloach [12], and Thomassen [16], for $k = 2$, and by Robertson and Seymour [9] for general but fixed k. With the line-graph reduction described by Perl and Shiloach in [8] the k-EDPP can also be solved in polynomial time. If we let α be the inverse Ackerman function as defined in [14], the currently best known time bounds for the k-DPP on undirected graphs, are $O(m\alpha(m, n) + n)$ for the 2-VDPP, $O(m\alpha(m, n) + n \log n)$ time for the 2-EDPP as shown by the author of this paper in [15], and $O(mn^2)$ time for the k-VDPP with fixed $k > 2$, and $O(m^2n^2)$ for the k-EDPP with fixed $k > 2$ as shown by Perković and Reed in [7].[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 [3]. However, in [8] 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 [3] generalized this result of Perl and Shiloach to an $O(mn^{k-1})$-time algorithm for the k-VDPP on dags for all $k \geq 2$. Lucchesi and Giglio [5] 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, such 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 [8] 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 [15] 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 [10] described an airplane routing problem that can be solved with an algorithm for the k-EDPP on dags.

New results. In some scenarios, given a tuple (s_1, s_2, t_1, t_2) of vertices, apart from testing whether there are two disjoint paths leading from the vertices

[1] For the last two results we also use the line graph reduction from the k-EDPP to the k-VDPP as well as a reduction from the decision version to the general version of the k-DPP that increases the running time by factor m.

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 without constructing such paths. 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 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, if such a tuple exists. This data structure can be constructed in $O((m+k)(\log_{2+(m+k)/(n+k)} n) + n \log^2 n)$ time.

As an application of this data structure and main result of this paper, extending some ideas of Lucchesi and Giglio [5] concerning a reduction for the decision version of the 2-VDPP, we show that it can be used to improve the running time for the 2-VDPP on dags from $O(mn)$ to $O(m(\log_{2+m/n} n) + n \log^3 n)$ time. Applying the reduction from the 2-EDPP to the 2-VDPP given in [15] results in an $O(m(\log_{2+m/n} n) + n \log^3 n)$ time algorithm for 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 $v \in T$ if $v \in V$ and $e \in T$ if $e \in E$. $f_T(v)$ denotes the father of v in T.

A *topological numbering* τ 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.

3 Finding Disjoint Paths Between Pairs of Vertices

Suurballe and Tarjan presented in [13] 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 τ is a topological numbering of the vertices of V, then, for each $v \in V$ with two edge-disjoint paths from s to v, $\tau(q(v)) < \tau(v)$ and $(p(v), v) \in E$.

2. If there are two edge-disjoint paths from s to v, then there are also two edge-disjoint paths from s to $q(v)$.

3. Two edge-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:

In a first round, mark v and, beginning in v with each marked vertex x, also mark $q(x)$ until reaching s. This process must stop because of property 1. In a second round p_1 is 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. Moreover, when visiting a marked vertex x, un-mark x. In a third round p_2 is constructed in the same way as p_1 following $(p(x), x)$ and un-marking x, if x is marked, and following $(f_T(x), x)$, if x is not marked.

Suurballe and Tarjan also observed that the construction of p_1 and p_2 unmarks all vertices marked in the first round of the construction. This guarantees that prior to the construction of a further pair of disjoint paths no vertex in our graph is marked. Note that we do not claim that property 3 follows immediately from the first two other properties. We only claim that the values $p(v)$ and $q(v)$ computed by the data structure of Suurballe and Tarjan have the above three properties.

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 $s \in V$ to a vertex $t \in V$ in G, if, and only if, there are two edge-disjoint paths from s_2 to t_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 p_1 and p_2 in $O(|p_1| + |p_2|)$ time.

In this section we want to show:

Lemma 1. *Let $G = (V, E)$ be a dag. Then, given $k+1$ pairs of vertices (s_1, s_2), $(u_1, v_1), \ldots, (u_k, v_k)$ it is possible to construct in $O((m+k)(\log_{2+(m+k)/(n+k)} n) + n \log^2 n)$ time a data structure that can output, for each pair (u_l, v_l) with $1 \le l \le k$ 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 exist. The paths themselves can be output in $O(|p_1| + |p_2|)$ time.*

In our proof of Lemma 1 disjoint means edge-disjoint, but with the previous reduction it also holds for vertex-disjoint paths.

Proof. Let $G' = (V', E')$ be the graph obtained by adding vertices $s, w_1, \ldots, w_k$ and 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 reduces to the problem of determining a data structure able to output, for each $i \in \{1, \ldots, k\}$, a tuple (s_1, y_1, s_2, y_2) such that there are disjoint paths p_1 and p_2 from s to w_i with p_j $(j \in \{1, 2\})$ using (s, s_j) as first and (y_j, w_i) as last edge, if such a tuple exists.

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 we define T, p, and q to be the shortest-path tree and the mappings constructed by this data structure with the properties 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 a lookup table containing the vertices $r_1(v)$ for all $v \in V$ (hence, $r_2(v)$ is the vertex $w \in \{s_1, s_2\}$ with $w \neq r_1(v)$). We start with a depth-first-search in T' and when visiting a vertex y, we colour the vertices of T such that all vertices $x \neq s$ on the tree path from s to y in T' are coloured black in T if $p_1(x)$ starts with edge (s, s_1), whereas, if $p_1(x)$ starts with edge (s, s_2), x is coloured red. All other vertices of T should be coloured white. In other words, if x is coloured black, we have $r_1(x) = s_1$, whereas, if x is coloured red, we have $r_1(x) = s_2$. Note that the red or black coloured vertices are exactly the vertices marked before the construction of $p_1(y)$ and $p_2(y)$.

Suppose our depth-first-search reaches a child y of s in T'. For constructing two disjoint paths from s to y with the data structure of Suurballe and Tarjan in the first round of the construction process only y and s are to be marked. Hence, it follows from the construction process described above that $r_1(y)$ is equal to y if $p(y) = s$, and equal to the first vertex $z \neq s$ on the tree path from s to $p(y)$ in T, if $p(y) \neq s$ (z can be determined in constant time if in a preprocessing step taking $O(m)$ time we determine for each $v \in T$ the first vertex $\neq s$ on the tree path from s to v in T). Hence, we know how to colour y correctly.

When reaching a vertex y not equal to a child of s in T', we will determine the last red or black coloured vertex $x \neq s$ before $p(y)$ on the tree path from s to $p(y)$ in T. Note that the ancestors of y in T' are exactly the vertices that would be marked by the data structure of Suurballe and Tarjan in the first round of constructing two disjoint paths from s to y and that all these nodes have already been coloured black or red by the depth-first-search in T'. If no red or black coloured vertex exists on the tree path from s to $p(y)$ in T, we know from the construction process of path $p_1(y)$, that $p_1(y)$ between s and $p(y)$ follows the tree path from s to $p(y)$ in T. Hence, y should be coloured black if (s, s_1) is the first edge on the path from s to $p(y)$ in T, and, if (s, s_2) is the first edge on this path, y should be coloured 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 (note that, if τ is a topological numbering of the vertices in G', the vertices that would be marked before the construction of two disjoint paths from s to x by the data structure of Suurballe and Tarjan are exactly the vertices v with $\tau(v) \leq \tau(x)$ that would by marked before the construction of disjoint paths from s to y and that $p_1(x)$ visits only vertices v with $\tau(v) \leq \tau(x)$). Hence, if by induction we have already shown that all ancestors of y in T' are coloured correctly, then y is also coloured correctly by colouring it in the same colour as x.

For the computation of the last coloured vertex x on the 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 coloured vertices from T_1, as well as all black coloured vertices from T_2. Let y' be the vertex that appears in the middle of the tree path from s to y in T (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 so, 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 colour 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 [4] for updating 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 our algorithm consists of $O(n)$ deletions of vertices and (adjacent) edges, $O(n)$ reinsertions, and only $O(n \log n)$ queries for determining the vertices $r_1(v)$ for all $v \in V$, the construction time of our data structure is bounded by $O((m+k)(\log_{2+(m+k)/(n+k)} n) + n \log^2 n)$. For each $v \in V$, $p_1(v)$ and $p_2(v)$ can be output with the data structure of Suurballe and Tarjan in $O(|p_1(v)| + |p_2(v)|)$ 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 dags. In the following, disjoint means always vertex-disjoint.

Let us call an instance $I = (G, s_1, s_2, t_1, t_2)$ of the 2-VDPP on a dag $G = (V, E)$ to be *irreducible* if the in-degree of each vertex $v \in V - \{s_1, s_2\}$ and the out-degree of each vertex $v \in V - \{t_1, t_2\}$ is at least two, and if t_1, t_2 have no outgoing and s_1, s_2 no incoming edges. On irreducible instances the following lemma holds:

Lemma 2. *Let (G, s_1, s_2, t_1, t_2) be an irreducible instance of the 2-VDPP on a dag $G = (V, E)$. Then, for each pair $v, w \in V$ with $v \neq w$, there are two disjoint paths p_1 and p_2 such that p_i $(1 \leq i \leq 2)$ leads from a vertex in $\{v, w\}$ to a vertex in $\{t_1, t_2\}$ as well as two disjoint paths leading from $\{s_1, s_2\}$ to $\{v, w\}$.*

Corollary 3 (Thomassen [17]). *If (G, s_1, s_2, t_1, t_2) is an irreducible instance of the 2-VDPP on a dag $G = (V, E)$, 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 paths is v.*

As observed by Thomassen [17], given an algorithm for solving the 2-VDPP on irreducible instances in $T(m, n)$ time, the 2-VDPP on dags can be solved in

$O(T(m,n)+m+n)$ time. Hence, in the following, we let $I = (G, s_1, s_2, t_1, t_2)$ be an irreducible instance of the 2-VDPP on a dag $G = (V, E)$. 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 $\{u, v\}$.

Let us first describe how the original algorithm of Lucchesi and Giglio finds two disjoint paths solving the 2-VDPP. In a first step it determines two disjoint paths p_1, from s_1 to t_1, and p_2, from s_2 to t_2, in $U(G)$. Like Lucchesi and Giglio, for two consecutive edges (u, v) and (v, w) on p_1 or p_2, let us refer to v as a *switch* if either both edges (u, v) and (w, v) are part of E (i.e. $(v, u), (v, w) \notin E$, since G is a dag), or $(v, u), (v, w) \in E$. Lucchesi and Giglio [5] proved that there is a choice of four vertices u, u', v, and v' in the following also called *boundary vertices* and of four paths r_1, r_2, q_1, q_2 in G such that p_1 and p_2 depending on the positions of u, u', v, v' can be replaced by one of the four pairs of paths given in the left column of Table 1 such that the resulting paths are disjoint (ignore the other columns of this table). Moreover, in Lucchesi's and Giglio's algorithm u can be chosen as the switch with the smallest and v as the switch with the largest topological number among all switches on p_1 and p_2. This guarantees that in each replacement of Table 1 replacing sub-paths of p_1 and p_2 by sub-paths of q_1 and q_2 the vertex u is not a switch of the new paths p_1^* and p_2^* and in all other cases v is not a switch of p_1^* and p_2^*. u' and v' are chosen in such a way that they are not switches of p_1 and p_2 neither before nor after the replacement. Being paths in G the paths q_1, q_2, r_1, and r_2 cannot contain any switch. Consequently, the set of switches of p_1^* and p_2^* is a proper subset of the switches of p_1 and p_2 before the replacement. Therefore, after $O(n)$ replacements as shown in Table 1 the resulting paths can no longer contain any switch and they solve the 2-VDPP. Since the running time for identifying the vertices u, u', v, and v' and the construction of the paths r_1, r_2, q_1 and q_2 is bounded by $O(m)$, Lucchesi's and Giglio's algorithm runs in $O(mn)$ time.

The main idea of the algorithm of this paper is to choose the vertices u, u', v, and v' much more carefully such that after each replacement at least a constant fraction of the switches of p_1 and p_2 are removed. This would reduce the number of replacements from $O(n)$ to $O(\log n)$. Unfortunately, this approach will not always be successful. In some sub-cases we will not be able to reduce the number of switches by a constant fraction. However, in all these cases using the data structure presented in Section 3 we will be able to guess the boundary vertices of the next sub-rounds without constructing the paths r_1, r_2, q_1, q_2. This will reduce the running time between two replacements which remove a constant fraction of switches to $O(m \log^2 n)$ time.

Let us now describe our new algorithm for the 2-VDPP on dags. Like Lucchesi and Giglio we start with the construction of two disjoint paths p_1, from s_1 to t_1, and p_2, from s_2 to t_2, in $U(G)$. This can be done in $O(m\alpha(m, n))$ time (cf. [15]). The remaining part of the algorithm is divided into several rounds.

Let us describe 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

Table 1. The path replacements 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.α	$u \in p_i[c_i, t_i], u' \in q_j$	$p_i^* := q_i[s_i, u] \circ p_i[u, t_i]$
1b.β	$u \in p_i[s_i, c_i), u' \in q_j$	$p_j := q_j[s_j, u'] \circ p_j[u', t_j]$
2b	$u \in p_i, u' \in q_j$	
1c.α	$u, v \in p_i, u' \in q_i, v' \in r_i, u \notin p_i(d_i, ti]$	$p_i^* := q_i[s_i, u'] \circ p_j[u', v'] \circ r_i[v', t_i]$
1c.β	$u, v \in p_i, u' \in q_i, v' \in r_i, u \in p_i(d_i, ti]$	$p_j^* := q_j[s_j, u] \circ p_i[u, v] \circ r_j[v, t_j]$
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	$u \in p_i, v \in p_j, u' \in q_i,\ v' \in r_j$	$p_j^* := q_j[s_j, u] \circ p_i[u, v'] \circ r_j[v', t_j]$

on p_i visited immediately before $(n_i - \lfloor \frac{1}{4} n_i \rfloor)$-th switch of p_i. If $\lfloor \frac{1}{4} n_i \rfloor = 0$, let $c_i := s_i, d_i := t_i$.

Let τ be a topological numbering of the vertices of G. Like Lucchesi and Giglio in [5] we define v to be the switch with largest topological number on p_1 and p_2, but unlike Lucchesi and Giglio we let v' be the first vertex x with $\tau(x) > \tau(v)$ on the path p_1 or p_2 not visiting v. We distinguish between 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, 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 $p_2[c_2, t_2]$ with the smallest topological number. In both cases we let u' be the last vertex x with $\tau(x) < \tau(u)$ on the path p_1 or p_2 not visiting u. We define q_1 and q_2 to be disjoint paths from s_1 and s_2 to u and u' such that q_i starts in s_i $(1 \leq i \leq 2)$, and, similarly, we let r_1 and r_2 be disjoint paths from v and v' to t_1 and t_2 such that r_i ends in t_i $(1 \leq i \leq 2)$. These paths exist because of Lemma 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 sub-cases of Case i. The new paths are disjoint:

Lemma 4. *p_1^* and p_2^* are disjoint.*

Proof. p_1^* and p_2^* are disjoint: For the Cases 1a, 2a, 1b.α, 1b.β, and 2.b this follows from the fact that the remaining 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.α, 1b.β, 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 each 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. The latter 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. □

After the path replacements of Table 1, u and v as vertices with the smallest or largest topological number can no longer be switches of p_1 or p_2. Unfortunately, u and v may be the only switches deleted from p_1 and p_2 in the Cases 1b.β, 1c.β, or 2a. Therefore, in these cases the idea is to consider not only one round but a series of k rounds such that in the first $k-1$ rounds we are in one of the Cases 1b.β, 1c.β, 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 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.β 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).[2]

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 Table 1 we can conclude that after each round (super-round in Case 1b.β, 1c.β, 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: Apart from u and v, in the Cases 1a, 1c.α, and 1d at least all switches of $p_1(d_1, t_1]$ or $p_2(d_2, t_2]$ and in the Cases 1b.α, 2b, 2c, and 2d at least all switches of $p_1[s_1, c_1)$ or $p_2[s_2, c_2)$ are removed (for the Cases 1d and 2d note that, as shown in the proof of Lemma 4, u appears before v' on p_i and u' before v on p_j). 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. Each round can be implemented efficiently:

Lemma 5. *Each round has a running time of* $O(m(\log_{2+m/n} n) + n \log^2 n)$.

Proof. For each round $n_1, n_2, c_1, c_2, d_1, d_2$ 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 s_1 and s_2 to u and u' as well as two disjoint paths from v and v' to t_1 and t_2 can be computed in $O(m)$ time. Given these paths, it is easy to decide in which case we are and to implement the path replacements for the Cases 1a, 1b.α, 1c.α, 1d, 2b, 2c, and 2d, again in $O(m)$ time.

We now consider the time complexity of the Cases 1b.β, 1c.β, and 2a. When talking about p_1 and p_2 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. 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 edges of p_1^1 and p_2^1 and in the j-th sub-round when replacing

[2] More precisely, if one of the vertices c_1, c_2, d_1, d_2 does no longer exist on these paths, we know that the replacement by which it was removed resulted in the deletion of at least a constant fraction of all switches from the paths given in the first sub-round and the corresponding sub-round can be defined as the last sub-round.

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 the paths given in this sub-round. For u^l and v^l ($1 \le l \le k$), this is true since all switches of p_1^l and p_2^l lie on the original part.

For $l \in \{1, \ldots, k\}$, let us define numbers a_l and b_l such that the a_l-th sub-round is the last sub-round before the l-th sub-round corresponding to Case 1b.β or 1c.β and the b_l-th sub-round is the last sub-round before the l-th sub-round corresponding to Case 2a or 1c.β (a_l or b_l 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 $u^{a_l}, u'^{a_l}, v^{b_l}$, and v'^{b_l}, 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 $\tau(u^{a_l}) \le \tau(x) \le \tau(v^{b_l})$ holds for all vertices $x \notin \{u'^{a_l}, v'^{b_l}\}$ on the original parts of p_1^l and p_2^l. Now, from $\tau(u'^{a_l}) < \tau(u^{a_l}) \le \tau(u^l) \le \tau(v^l) \le \tau(v^{b_l}) < \tau(v'^{b_l})$ we can conclude that the vertices u'^l and v'^l must appear after a vertex $x \in \{u^{a_l}, u'^{a_l}\}$ on p_1^l or p_2^l or be equal to x and they must appear before a vertex $y \in \{v^{b_l}, v'^{b_l}\}$ 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 always lie on the original part of p_1 or p_2 which always is a sub-path 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 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 repeatedly delete the vertex with the largest topological number from this list until we find a vertex x lying on the original part of p_1 or p_2. We 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, similarly, this also holds for the boundary vertex u.

We now describe the computation of u'^l and v'^l for $1 \le l \le k$: For each $i \in \{1, 2\}$, let us number the vertices visited by the paths p_i^1 in the order in which they appear on p_i^1 and let us construct a list of all switches of p_i^1 sorted by these numbers. Using these lists we can identify the last switch of p_i^l by a binary search in $O(\log n)$ time. If $v^l \in p_j^l$ holds for $j \in \{1, 2\}$ with $j \ne i$, then v'^l is the first vertex x with a topological number larger than that of v^l on the part of p_i^l between the last switch of p_i^l and the endpoint of the original part of p_i^l appearing after the last switch. Since the vertices on this part of p_i^l are sorted by their topological numbers, v'^l can be determined by a further binary search again in $O(\log n)$ time. Hence, the time needed for the construction of the boundary vertices v'^l for all sub-rounds can be bounded by $O(n \log n)$ time and, similarly, this also holds for the computation of the boundary vertices u'^l.

Therefore, if we know for each sub-round which case is applicable, i.e. if we know the original parts of p_1^l and p_2^l of the following sub-round, we can efficiently compute u^l, u'^l, v^l, and v'^l for each sub-round. In order to determine the relevant case, 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.β, 1c.β, or 2a, we assume that we are in this case and, under this assumption, we compute the boundary vertices of 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 $i, j \in \{1, 2\}$ we assume that we are in Case 1b.β (note that we will never encounter more than one of the Cases 1b.β, 1c.β, and 2a).

After the first phase we construct in maximal $O((m+n)(\log_{2+(m+n)/2n} n) + n^2 \log n) = O(m(\log_{2+m/n} n) + n^2 \log n)$ time 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) 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 q_1, from s_1 to w_1, and q_2, from s_2 to w_2, and in a similar way again using the data structure of Lemma 1 we can construct a tuple (x_1, t_1, x_2, t_2) with $\{x_1, x_2\} = \{v^l, v'^l\}$ such that there are two disjoint paths r_1, from x_1 to t_1, and r_2, from x_2 to t_2. We finally test whether we are in one of the Cases 1b.β, 1c.β, 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 we must be in the last sub-round of the super-round.

Concerning the paths p_1^{k+1} and p_2^{k+1} resulting from the last sub-round of our super-round, if in the last sub-round we are in one of the c- or d-Cases of Table 1, they consist of three pairs of disjoint paths: q_1 and q_2, from s_1 and s_2 to u^k and u'^k, r_1 and r_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 determine 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 an a- or b-Case, we can construct p_1^{k+1} and p_2^{k+1} in $O(n)$ time. For details see the full version of this paper. ☐

Theorem 6. *On dags the 2-VDPP is solvable in* $O(m(\log_{2+m/n} n) + n \log^3 n)$ *time.*

Proof. In a first step we reduce the problem to a dag G with $O(n)$ edges:

Lucchesi and Giglio [5] have shown that two disjoint paths from s_1 to t_1 and from s_2 to t_2 on a dag $G = (V, E)$ can be constructed from two disjoint paths p_1 and p_2 in $U(G)$ by replacing sub-paths of p_1 and p_2 by sub-paths of a set S of paths. 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 of two disjoint paths from v to y for every $v \in V$. Such paths must exist because of Corollary 3.

If we choose as paths from x to vertices $v \in V$ the paths that would be constructed by the data structure of Suurballe and Tarjan [13], these paths visit only edges of the shortest-path tree T and edges of the form $(p(w), w)$ with T and p being defined as in the beginning of Section 3. Consequently, the graph containing these $O(n)$ edges plus $O(n)$ edges for the construction of disjoint paths from vertices $v \in V$ to y, as well as 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.

The running time for the reduction of our problem to a sparse graph with only $O(n)$ edges is dominated by the construction time of $O(m(\log_{2+m/n} n)+n\log^2 n)$ for the data structure of Suurballe and Tarjan. After the reduction two disjoint paths on $U(G)$ can be computed in $O(n\alpha(n,n))$ time [15]. The following $O(\log n)$ rounds run in $O(n\log^2 n)$ time (Lemma 5). □

References

1. G. Di Battista, R. Tamassia, and L. Vismara, Output-sensitive reporting of disjoint paths, *Algorithmica* **23** (1999), pp. 302–340.
2. Y. Dinitz and J. Westbrook, Maintaining the classes of 4-edge-connectivity in a graph on-line, *Algorithmica* **20** (1998), pp. 242–276.
3. S. Fortune, J. Hopcroft, and J. Wyllie, The directed subgraph homeomorphism problem, *Theoret. Comput. Sci.* **10** (1980), pp. 111–121.
4. 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.
5. 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.
6. T. Ohtsuki, The two disjoint path problem and wire routing design, Proc. Symposium on Graph Theory and Algorithms, Lecture Notes in Computer Science, Vol. 108, Springer, Berlin, 1981, pp. 207–216.
7. 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. 365–371.
8. Y. Perl and Y. Shiloach, Finding two disjoint paths between two pairs of vertices in a graph, *J. ACM* **25** (1978), pp. 1–9.
9. N. Robertson and P. D. Seymour, Graph minors. XIII. The disjoint paths problem, *J. Comb. Theory, Ser. B*, **63** (1995), pp. 65–110.
10. A. Schrijver, A group-theoretical approach to disjoint paths in directed graphs, *CWI Quarterly* **6** (1993), pp. 257–266.
11. P. D. Seymour, Disjoint paths in graphs, *Discrete Math.* **29** (1980), pp. 293–309.
12. Y. Shiloach, A polynomial solution to the undirected two paths problem, *J. ACM* **27** (1980), pp. 445–456.
13. J. W. Suurballe and R. E. Tarjan, A quick method for finding shortest pairs of disjoint paths. *Networks* **14** (1984), pp. 325-336.
14. R. E. Tarjan and J. van Leeuwen, Worst-case analysis of set union algorithms, *J. ACM* **31** (1984), pp. 245–281.
15. 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.
16. C. Thomassen, 2-linked graphs, *Europ. J. Combinatorics* **1** (1980), pp. 371–378.
17. C. Thomassen, The 2-linkage problem for acyclic digraphs, *Discrete Math.* **55** (1985), pp. 73–87.