Solving the 2-Disjoint Paths Problem in Nearly Linear Time

Torsten Tholey

2004

Abstract

Given four distinct vertices s_1, s_2, t_1, and t_2 of a graph G, the 2-disjoint paths problem is to determine two disjoint paths, p_1 from s_1 to t_1 and p_2 from s_2 to t_2, if such paths exist. Disjoint can mean vertex- or edge-disjoint.

Both, the edge- and the vertex-disjoint version of the problem, are $\mathcal{NP}$-hard in the case of directed graphs. For undirected graphs, we show that the $O(mn)$-time algorithm of Shiloach can be modified so as to solve the 2-(vertex-)disjoint paths problem in $O(n+m\alpha(m,n))$ time, where m is the number of edges in G, n is the number of vertices in G, and α denotes the inverse of the Ackermann function. Our result also improves the running time for the 2-edge-disjoint paths problem on undirected graphs as well as the running times for the decision versions of the 2-vertex- and the 2-edge-disjoint paths problem on dags.

Solving the 2-Disjoint Paths Problem in Nearly Linear Time

Torsten Tholey

Institut für Informatik
Johann Wolfgang Goethe-Universität Frankfurt
D-60054 Frankfurt am Main, Germany
tholey@cs.uni-frankfurt.de

Abstract. Given four distinct vertices s_1, s_2, t_1, and t_2 of a graph G, the 2-disjoint paths problem is to determine two disjoint paths, p_1 from s_1 to t_1 and p_2 from s_2 to t_2, if such paths exist. Disjoint can mean vertex- or edge-disjoint.
Both, the edge- and the vertex-disjoint version of the problem, are $\mathcal{NP}$-hard in the case of directed graphs. For undirected graphs, we show that the $O(mn)$-time algorithm of Shiloach can be modified so as to solve the 2-(vertex-)disjoint paths problem in $O(n + m\alpha(m, n))$ time, where m is the number of edges in G, n is the number of vertices in G, and α denotes the inverse of the Ackermann function. Our result also improves the running time for the 2-edge-disjoint paths problem on undirected graphs as well as the running times for the decision versions of the 2-vertex- and the 2-edge-disjoint paths problem on dags.

1 Introduction

In the k-disjoint paths problem ($k \in \mathbb{N}$) we have $2k$ pairwise distinct vertices $s_1, \ldots, s_k, t_1, \ldots, t_k$ of a graph $G = (V, E)$, and we want to output k pairwise disjoint paths p_i from s_i to t_i ($1 \leq i \leq k$), if such paths exist. For short, we will subsequently refer to the k-disjoint paths problem as k-DPP or, more precisely, as k-VDPP if disjoint means vertex-disjoint, and as k-EDPP if disjoint means edge-disjoint. For time bounds, we let $n = |V|$ and $m = |E|$.

The k-DPP arises in the context of VLSI-design, routing problems, and network reliability (see [1] and [15]) and has been extensively studied. A short overview is given in this introduction. Further overviews can be found in [2], [21], and [22]. We will also consider the *decision version* of the k-DPP in which we only want to test the existence of k disjoint paths p_i from s_i to t_i ($1 \leq i \leq k$). Given an $O(T(n, m))$-time algorithm for the decision version of the k-DPP, the disjoint paths, if they exist, can be computed within $O(n+mT(m, n))$ time using the following algorithm: Step through all edges of the input graph G, and, for each edge e considered, if there are k pairwise disjoint paths connecting s_i to t_i for $1 \leq i \leq k$ in the graph $G - \{e\}$,[1] delete e from G before considering the next

[1] For short, given a graph $G = (V, E)$ and a set $W \subseteq V$ we define $G - W$ to be the graph $(V - W, E \cap \{\{u, v\} \mid u, v \in V - W\})$, and, similarly, for each set $F \subseteq E$ we let $G - F$ be the graph $(V, E - F)$.

edge. After having considered all edges the resulting graph consists of exactly k disjoint paths connecting s_i to t_i for $1 \leq i \leq k$. The paths themselves can be output with a depth-first search in the case of the k-VDPP. In the case of the k-EDPP the construction of the paths is more complicated. It can be done in $O(n + mT(n, m))$ time, but we will not show this here.

Previous results. 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 [5]. However, in [17] Perl and Shiloach presented an $O(mn)$-time algorithm for solving the 2-VDPP on dags (directed acyclic graphs). Fortune, Hopcroft, and Wyllie [5] generalized this result of Perl and Shiloach to a polynomial-time algorithm for the k-VDPP on dags. Lucchesi and Giglio [13] gave a linear-time reduction from the decision version of the 2-VDPP on dags to the 2-VDPP on undirected graphs. Solving the latter problem with the previously best known algorithm for the 2-VDPP on undirected graphs leads to an improved running time of $O(n^2)$ for the decision version of the 2-VDPP on dags. This bound holds also for the decision version of the 2-EDPP as we will show in Sect. 3 of this paper by using an $O(n + m \log_{2+m/n} n)$-time reduction from the 2-EDPP to the 2-VDPP. As an application Schrijver [20] described an air plane routing problem that can be solved with an algorithm for the k-EDPP on dags.

If k is not fixed, as defined above, but part of the input, the problem of testing whether there are k disjoint paths p_i from s_i to t_i $(1 \leq i \leq k)$ is $\mathcal{NP}$-complete also for undirected graphs. This was shown by Knuth, cf. [10], and Lynch [14] for the VDPP and by Even, Itai, and Shamir [4] for the EDPP.

The first polynomial-time algorithms for the 2-DPP on undirected graphs were given by Ohtsuki [15], Seymour [23], Shiloach [24], and Thomassen [26]. More precisely, Seymour gave only a solution for the decision version of the 2-DPP, but for both, the 2-EDPP and the 2-VDPP. Ohtsuki, Shiloach, and Thomassen considered only the 2-VDPP. However, with the reductions described in [17] their algorithms can also be used for solving the 2-EDPP without increasing the running time of $O(nm)$ of Ohtsuki's and Shiloach's algorithms. Later, Khuller, Mitchell, and Vazirani implicitly showed in [11] that the algorithm of Shiloach can be modified so as to run in $O(n^2)$ time. Using an appropriate reduction, one can also show that the 2-EDPP can be solved within the same time bound (see Sect. 5 for an example of such a reduction). Finally, in [7] Gustedt described an $O(n + m \log n)$-time algorithm for the 2-VDPP on undirected graphs. Unfortunately, since some of the lemmas in [7] fail for certain types of graphs, the current version of Gustedt's algorithm does not work on all graphs. But, for all instances of the 2-VDPP for which G is a triconnected graph such that there is no vertex v of G with $v \notin \{s_1, s_2, t_1, t_2\}$ that can be separated from $\{s_1, s_2, t_1, t_2\}$ by a separator of size three, the lemmas of [7] mentioned above hold and, as a byproduct of this paper, we prove that the 2-VDPP can be reduced to the 2-VDPP on this restricted set of graphs. Concerning the more general k-DPP, Robertson and Seymour [18] showed that the decision version of the undirected k-VDPP is solvable in $O(n^3)$ time. Finally, Perković and Reed [16] improved the running time to $O(n^2)$, which is currently the best known time

bound for the decision version of the k-DPP on undirected graphs, for all $k \geq 3$. For some kinds of graphs, the 2-DPP or the general k-DPP are solvable in linear time (see [17], [6], [12], and [19]).

New results. In this paper we present an algorithm for solving the 2-VDPP on undirected graphs in $O(n + m\alpha(m, n))$ time, which improves the previously best known time bound of $O(n^2)$ to $O(\min\{n^2, n + m\alpha(m, n)\})$. α denotes the inverse of the Ackermann function. Applying some well-known reductions or slightly modified versions thereof, we will also show that our result also leads to an $O(m\alpha(m, n) + n \log n)$ time bound for the 2-EDPP on undirected graphs, an $O(n + m\alpha(m, n))$ time bound for the decision version of the 2-VDPP on dags, and finally an $O(n + m \log_{2+m/n} n)$ time bound for the decision version of the 2-EDPP on dags.

The paper is organized as follows. In Sect. 2 we discuss Shiloach's algorithm for the 2-VDPP on undirected graphs. We will see that the 2-VDPP can be reduced to a problem that basically consists in eliminating, without changing the solvability of the problem, all vertices of an instance of the 2-VDPP that can be separated from s_1, s_2, t_1, and t_2 by a separator consisting of three or fewer vertices. Following the presentation, in Sect. 3, of some simple definitions concerning the k-vertex-connectivity, the problem above is solved in Sect. 4. Other versions of the 2-DPP are discussed in Sect. 5.

2 Shiloach's Algorithm

In this section we consider Shiloach's algorithm for the 2-VDPP on undirected graphs. For an instance $I = (G, s_1, s_2, t_1, t_2)$ of the 2-VDPP, if there exist two vertex-disjoint paths p_1 from s_1 to t_1 and p_2 from s_2 to t_2, we say that I *has a solution* and that p_1 *and* p_2 *solve* I. Given two instances $I_1 = (G, s_1, s_2, t_1, t_2)$ with $G = (V, E)$ and $I_2 = (G', s_1', s_2', t_1', t_2')$ with $G' = (V', E')$ of the 2-VDPP, let us say that I_1 is *2-paths reducible* (or for short *2P-reducible*) to I_2 if the following conditions hold: First, I_1 has a solution iff I_2 has a solution; second, $|V'| = O(|V|)$; third, $|E'| = O(|E|)$; and finally, fourth, given a solution of I_2 we can solve I_1 in $O(|V| + |E|)$ time. If we replace an instance I_1 of the 2-VDPP by another instance I_2 such that I_1 is 2P-reducible to I_2 we also say that I_1 is *2P-reduced* to I_2.

In [24] Shiloach outlined a proof of Itai according to which we can assume w.l.o.g. that the input graph $G = (V, E)$ of an instance $I = (G, s_1, s_2, t_1, t_2)$ of the 2-VDPP is triconnected. On a triconnected graph the algorithm of Shiloach proceeds as follows: If G is planar, the problem instance I is solved with the algorithm of Perl and Shiloach [17]. Otherwise I is 2P-reduced to an instance $I' = (G', s_1, s_2, t_1, t_2)$ of the 2-VDPP with $G' = (V', E')$ a triconnected graph such that there are four vertex-disjoint paths from s_1, s_2, t_1, and t_2 to any other set $S \subseteq V' - \{s_1, s_2, t_1, t_2\}$ with at most four vertices (if $|S| < 4$ the end points of the paths in S may overlap). According to Shiloach, the 2-VDPP on such an instance $I' = (G', s_1, s_2, t_1, t_2)$ can be solved as follows: First extract a subgraph of G' homeomorphic to K_5 or $K_{3,3}$. If no such subgraphs exist, G' is planar and

I' can be solved with the algorithm of Perl and Shiloach [17]. Otherwise, if there is a subgraph homeomorphic to K_5, two disjoint paths from s_1 to t_1 and from s_2 to t_2 can be found with an algorithm of Watkins [27]. For the remaining case, Shiloach proved that using some further reductions two disjoint paths from s_1 to t_1 and from s_2 to t_2 can be constructed in linear time.

Concerning the complexity of the algorithm above, Shiloach showed that nearly all steps of the algorithm run in $O(m+n)$ time. Only two bottlenecks were not solved in $O(m+n)$ time by Shiloach: The first one is the reduction from I to I', and the second one is the extraction of a subgraph homeomorphic to $K_{3,3}$ or K_5. But Williamson [28] gave a linear-time algorithm for the latter problem. Thus, for solving the 2-VDPP in $O(n+m\alpha(m,n))$ time, we only need to show that, given an instance $I=(G,s_1,s_2,t_1,t_2)$ of the 2-VDPP, where G is a triconnected graph, it is possible to construct, in $O(m\alpha(m,n))$ time, an instance $I'=(G',s_1,s_2,t_1,t_2)$ of the 2-VDPP, where $G'=(V',E')$ is a triconnected graph such that in G' there are four disjoint paths from s_1, s_2, t_1, and t_2 to any other set $S \subseteq V' - \{s_1,s_2,t_1,t_2\}$ with $|S| \leq 4$, and I is 2P-reducible to I'. This is exactly what we will do in Sect. 4.

3 k-Vertex-Connectivity – Facts and Definitions

The following facts and definitions related to k-vertex-connectivity will be useful for understanding the correctness of our algorithm for the 2-VDPP in Sect. 4. Let $G=(V,E)$ be an undirected graph and let $s,t \in V$ with $s \neq t$. We say that s and t are *k-vertex-connected* (or that s is k-vertex-connected to t), iff there are k internally vertex-disjoint paths from s to t. By internally vertex-disjoint we mean that every pair of the k disjoint paths has no vertex in common, except s and t. We say that G is *k-vertex-connected* if all pairs of vertices of G are k-vertex-connected. If two distinct vertices s and t of a graph $G=(V,E)$ with $\{s,t\} \notin E$ are not $(k+1)$-vertex-connected, they can be separated by a k-separator:

Definition 1. *Let $G=(V,E)$ be an undirected graph. Then we call a subset $S \subseteq V$ with $|S|=k$ a (k-)separator (of G) iff the graph $G^*=G-S$ is not connected. If two vertices s,t of G^* are not connected in G^*, we say that S separates s and t (in G). If, for a vertex s and a set $T \subseteq V$, the connected component of G^* containing s does not contain any vertex of T, we say that s can be separated from T by S (in G) or that S separates s from T (in G).*

For an undirected graph $G=(V,E)$ and two distinct vertices $s,t \in V$, let us define $\kappa(s,t)$ as the size of a smallest separator S (containing neither s nor t) that separates s and t if $\{s,t\} \notin E$, and as one plus the size of a smallest separator S (containing neither s nor t) that separates s and t in $G-\{\{s,t\}\}$ if $\{s,t\} \in E$. Then there is an alternative characterization of k-vertex-connectivity:

Lemma 2. *Two vertices s and t of an undirected graph $G=(V,E)$ are k-vertex-connected iff $\kappa(s,t) \geq k$. G is k-vertex-connected iff $\kappa(s,t) \geq k$ for all $s,t \in V$.*

One can also show:

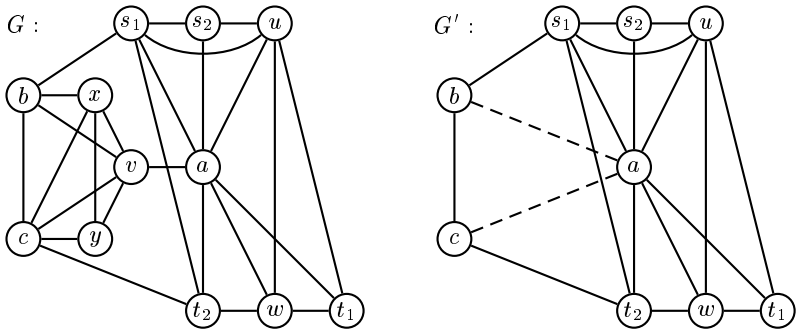

Fig. 1. A graph G and the Δ-replacement G' of G by the triangular cut $S = \{a, b, c\}$. The dashed lines represent newly inserted edges.

Lemma 3. *Let S be a k-separator that separates two vertices v and w of an undirected graph G and let $p_1, p_2, \ldots, p_k$ be k internally vertex-disjoint paths from v to w. Then, every path p_i with $1 \leq i \leq k$ contains exactly one vertex of S.*

In the following, we use "biconnected" and "triconnected" as synonyms for "2-vertex-connected" and "3-vertex-connected", respectively. Moreover, queries, referred to as k-connectivity queries, ask whether two given vertices v and w are k-vertex-connected. Finally, in the following section, disjoint will always mean vertex-disjoint and k-connected will mean k-vertex-connected.

4 An $O(n + m\alpha(m, n))$-time algorithm for the 2-VDPP

As we have already seen in Sect. 2, for solving the 2-VDPP in $O(n + m\alpha(m, n))$ time on undirected graphs, we only need to show that, given an instance $I = (G, s_1, s_2, t_1, t_2)$ of the 2-VDPP, where G is a triconnected graph, it is possible to construct, in $O(m\alpha(m, n))$ time, an instance $I' = (G', s_1, s_2, t_1, t_2)$ of the 2-VDPP, where $G' = (V', E')$ is a triconnected graph such that there are four disjoint paths from s_1, s_2, t_1, and t_2 to any other set $S \subseteq V' - \{s_1, s_2, t_1, t_2\}$ with at most four vertices (except that the end points in S may overlap), and I is 2P-reducible to I'. For this we will make use of so-called triangle-replacements:

Suppose we are given an instance $I = (G, s_1, s_2, t_1, t_2)$ of the 2-VDPP such that there exists a vertex v in G that is separated from $\{s_1, s_2, t_1, t_2\}$ by a 3-separator $S = \{a, b, c\}$. Then, like Gustedt in [7], we call S a *triangular cut of* G and the graph G' obtained from G by deleting all vertices of the connected component of $G - S$ containing v together with their adjacent edges and by inserting edges between all pairs of vertices of S that are not already connected by an edge the *triangle-replacement of G by S (removing vertex v)* or, for short, the *Δ-replacement of G by S* (see Fig. 1). Sometimes we also call the replacement step that replaces G by G' a Δ-replacement.

Shiloach [24] showed that if G contains no triangular cut separating a vertex v from $\{s_1, s_2, t_1, t_2\}$, then there are four disjoint paths from s_1, s_2, t_1, and t_2 to any other set $S \subseteq V - \{s_1, s_2, t_1, t_2\}$ with at most four vertices. Thus, if for an instance $I = (G, s_1, s_2, t_1, t_2)$ the graph G does not contain a triangular cut, we already know that we can solve the 2-VDPP in linear time. Otherwise, we just eliminate all triangular cuts from G. More precisely, we repeatedly search for a vertex v of G that is separated from $\{s_1, s_2, t_1, t_2\}$ by a triangular cut S and replace I by $I^* = (G^*, s_1, s_2, t_1, t_2)$, where G^* is the Δ-replacement of G by S removing v. We stop if, for the resulting instance $I' = (G', s_1, s_2, t_1, t_2)$ after all Δ-replacements, G' has no triangular cut S separating a vertex v from $\{s_1, s_2, t_1, t_2\}$.

One can easily show that the Δ-replacements do not change the solvability of the problem (see [24] for example).

Lemma 4. *Let $I = (G, s_1, s_2, t_1, t_2)$ be an instance of the 2-VDPP such that G is a triconnected graph and let G^* be a Δ-replacement of G. Then G^* is also a triconnected graph and $I^* = (G^*, s_1, s_2, t_1, t_2)$ has a solution iff I has a solution.*

Moreover, Shiloach [24] showed that if I is our original instance before the first Δ-replacement and I' is the instance of the 2-VDPP after the last Δ-replacement, then the following holds:

Lemma 5. *I is $2P$-reducible to I'. In particular, given two paths p'_1 and p'_2 that solve I', one can construct, in linear time, two paths p_1 and p_2 that solve I.*

Concerning the running time of the reduction from I to I' one can show:

Lemma 6. *Given an instance $I = (G, s_1, s_2, t_1, t_2)$ of the 2-VDPP, where $G = (V, E)$ is triconnected graph, a triangular cut S of G, and a vertex $v \in V$ that is separated from $\{s_1, s_2, t_1, t_2\}$ by S, the Δ-replacement $G^* = (V^*, E^*)$ of G by S removing v can be constructed in $O(|E - E^*|)$ time.*

Proof. Just start a depth-first search on $G - S$ with v as the source node and stop after the depth-first search tree T with root v is complete. Then G^* is the graph obtained from G by deleting all vertices of T and all their adjacent edges in G and by inserting a constant number of edges between vertices of S, if necessary. Since the set of vertices of T is a subset of $V - V^*$, it is easy to see that the construction of T and the deletion of the vertices and edges from G can be done in $O(|E - E^*|)$ time. □

From Lemma 6 we can conclude that, beside the time needed for finding a vertex v that can be separated from $\{s_1, s_2, t_1, t_2\}$ by a triangular cut and the time needed for finding such a triangular cut, the construction of our final instance I' without any triangular cut takes linear time. Thus, we just need to search for an efficient algorithm for determining a vertex v and a triangular cut S such that v is separated from $\{s_1, s_2, t_1, t_2\}$ by S. The main difference between the algorithm of this paper and Shiloach's algorithm is the computation of such vertices and triangular cuts. While not searching explicitly for triangular cuts

Shiloach's algorithm is occasionally unable to proceed due to the presence of such a cut. The cut is then removed by a Δ-replacement. After the Δ-replacement Shiloach's algorithm is restartet on the resulting graph. Since this my happen $\Theta(n)$ times, the running time of Shiloach's algorithm is bounded by only $O(mn)$.

Here, in contrast, we systematically remove all triangular cuts separating a vertex v from $\{s_1, s_2, t_1, t_2\}$ efficiently. For identifying such a vertex v, the following lemma will be very useful.

Lemma 7. *Let $G = (V, E)$ be an undirected triconnected graph containing four vertices s_1, s_2, t_1, and t_2, and let $G_x = (V_x, E_x)$ be the graph that is obtained from G by adding a new vertex x to V and new edges $\{x, s_1\}, \{x, s_2\}, \{x, t_1\}$, and $\{x, t_2\}$ to E. Then, a vertex $v \in V - \{s_1, s_2, t_1, t_2\}$ can be separated from $\{s_1, s_2, t_1, t_2\}$ by a set $S \subseteq V$ with $|S| \leq 3$ in G iff S separates v and x in G_x.*

Proof. Every path from v to x in G_x must visit a vertex $w \in \{s_1, s_2, t_1, t_2\}$ before reaching x. Hence, if S is a 3-separator separating v from $\{s_1, s_2, t_1, t_2\}$ in G, every path from v to x must visit a vertex in S before (or exactly when) visiting a vertex in $\{s_1, s_2, t_1, t_2\}$. Thus, S separates v and x in G_x.

Conversely, if S is a 3-separator separating x and v in G_x, then the vertices of $\{s_1, s_2, t_1, t_2\}$ are contained either in S or in the connected component of $G_x - S$ containing x. It follows that the connected component of $G_x - S$ containing v does not contain any vertex of $\{s_1, s_2, t_1, t_2\}$ and that the same is true of the graph $G - S$. Hence, v is separated from $\{s_1, s_2, t_1, t_2\}$ in G by S. □

Now, for an instance $I = (G, s_1, s_2, t_1, t_2)$ of the 2-VDPP, like in Lemma 7, let G_x be the graph obtained from G, by adding a new vertex x and edges $\{x, s_1\}$, $\{x, s_2\}$, $\{x, t_1\}$, and $\{x, t_2\}$ to G. Then, if G is triconnected, the same is true of G_x, and s_1, s_2, t_1, t_2 are 4-connected to x. Lemma 7 implies that in each reduction step, replacing an instance $I = (G, s_1, s_2, t_1, t_2)$ by an instance $I^* = (G^*, s_1, s_2, t_1, t_2)$ such that G^* is a Δ-replacement of G, for identifying a vertex $v \notin \{s_1, s_2, t_1, t_2\}$ that can be separated from $\{s_1, s_2, t_1, t_2\}$ by a triangular cut, we only need to look for a vertex v that is not 4-connected to x in G_x (note that v is not adjacent to x). To find such a vertex we could step through all vertices of V and test, for each vertex, whether it is 4-connected to x (we will later see efficient implementations for answering 4-connectivity queries). But, if we recompute the set of all vertices that are not 4-connected to x after each update of G, we might have to answer up to $\Omega(n^2)$ connectivity queries for the whole sequence of all reduction steps. The following lemma will help us to reduce the number of connectivity queries.

Lemma 8. *Let $I = (G, s_1, s_2, t_1, t_2)$ be an instance of the 2-VDPP such that $G = (V, E)$ is a triconnected graph, let S be a triangular cut of G, let v be a vertex of G that is separated from $\{s_1, s_2, t_1, t_2\}$ by S, and, finally, let G^* be the Δ-replacement of G by S removing v. Then, if two distinct vertices $a \in V^* - S$ and $b \in V^*$ are 4-connected in G, they are also 4-connected in G^*.*

Proof. Let $a \in V^* - S$, $b \in V^*$ be two distinct vertices that are 4-connected in G, and let p_1, p_2, p_3, and p_4 be pairwise internally vertex-disjoint simple paths connecting a and b.

If $b \in V^* - S$, only one path p out of our four paths can visit a vertex in $V - V^*$, and such a path must visit at least two vertices of S. Let us define c to be the first vertex on p in S, and d to be the last vertex on p in S. Now, if we replace the sub-path of p from c to d by the edge $\{c, d\}$, p and the other three paths of p_1, p_2, p_3, and p_4 are four internally vertex-disjoint paths in G^*.

If $b \in S$, no more than two of the paths p_1, p_2, p_3, and p_4 can visit a vertex in $V - V^*$. If they do so, they also visit a vertex in $S - \{b\}$. Then we follow these paths up to a first node $c \in S - \{b\}$, and then use the edge $\{c, b\}$ to reach b. After this replacement, the four paths are pairwise internally vertex-disjoint in G^*. □

Hence, having replaced G by a new graph G^*, if we search for a vertex v that is not 4-connected to x, we can exclude all vertices for which, in a previous reduction step, we have already tested whether they are 4-connected to x. If such a vertex was 4-connected to x, it remains 4-connected to x, and, otherwise, it was deleted from G. Thus, the number of 4-connectivity-queries is reduced to $O(n)$.

For efficiently supporting 4-connectivity queries, we might consider using a *dynamic data structure*. This is a data structure that supports two kinds of operations: update operations, which in case of graph problems usually consist of edge insertions and edge deletions, and queries, which in our case will be 4-connectivity queries. The idea behind dynamic data structures for graph problems is that, using the knowledge about a graph before an edge insertion or edge deletion, possibly queries can be answered faster than without such knowledge. Unfortunately, there is no known dynamic data structure supporting all of the operations above in nearly constant time. However, Kanevsky, Tamassia, Di Battista, and Chen [9] presented a very efficient *incremental dynamic data structure* supporting only edge insertions and 4-connectivity queries. This data structure can be initialized, in $O(m\alpha(m, n))$ time, with a triconnected graph containing m edges and n vertices, and, after this initialization, it supports $O(m)$ queries and insertions in $O(m\alpha(m, n))$ time. With a simple trick we can make use of this data structure: In addition to G_x, we also maintain a graph H_x. that, before the first Δ-replacement, is initialized with a copy of G_x. In a reduction step replacing G_x by a Δ-replacement G_x^* of G, we do not delete any vertex or edge of H_x, but insert in H_x the same edges that are inserted in G_x. Now, if we want to test whether a vertex of G_x is 4-connected to x in G_x, we only need to ask whether it is 4-connected to x in H_x, as shown by Lemma 9. Queries in H_x can now be answered with the data structure of Kanevsky et al..

Lemma 9. *A vertex w of G_x is 4-connected to x in G_x iff it is 4-connected to x in H_x.*

Proof. Let us define a set of k-paths $p_1, p_2, \ldots, p_k$ to be *quasi internally disjoint* (or, for short, *q.i. disjoint*) if, for all pairs (i, j) with $i, j \in \{1, \ldots, k\}$ and $i \neq j$, no inner vertex of p_i appears on p_j. We first show that, before and after each reduction step – i.e. Δ-replacement of G_x – transforming $G_x = (V_{G_x}, E_{G_x})$ and $H_x = (V_{H_x}, E_{H_x})$ into new graphs, the following invariant holds: If, in H_x, there are k q.i. disjoint simple paths p_i $(1 \leq i \leq k)$ connecting x to a vertex

$v_i \in V_{H_x} \cap V_{G_x}$, where $v_1, v_2, \ldots, v_k$ need not necessarily be distinct, there are also k q.i. disjoint paths q_i $(1 \leq i \leq k)$ connecting x to v_i in G_x such that the set of the vertices visited by q_i is a subset of the vertices visited by p_i. The invariant holds before the first reduction step, since H_x is initialized with G_x. Let us now assume that, for an $l \in \mathbb{N}$, after l reduction steps, there are k q.i. disjoint paths p_i $(1 \leq i \leq k)$ in H_x connecting x to a vertex $v_i \in V_{H_x} \cap V_{G_x}$. Moreover, let S be the triangular cut used for the first Δ-replacement of our algorithm, i.e. in the first reduction step of our algorithm. If one path p of the paths $p_1, \ldots, p_k$ uses an edge $\{a, b\}$ that is deleted from G_x after the first Δ-replacement, this path must also visit a vertex $c \in S$ before following edge $\{a, b\}$ and a vertex $d \in S$ after having reached edge $\{a, b\}$. We can now replace the sub-path of p between c and d by edge $\{c, d\}$. It is easy to see that, after this replacement, the paths $p_1, \ldots, p_k$ remain pairwise q.i. disjoint. We eventually must repeat this step to obtain k pairwise q.i. disjoint paths $p'_1, \ldots, p'_k$ such that no path uses an edge that is deleted from G_x after the first Δ-replacement, and such that the vertices visited by p'_i are a subset of the vertices visited by p_i. In the same way, we can replace all edges of any path in $p'_1, \ldots, p'_k$ that are deleted from G_x after the second, third, ... Δ-replacement until the resulting paths use only edges that are not deleted from G_x after l reduction steps.

The invariant above implies that, if, after an arbitrary number of reduction steps, x is 4-vertex-connected to a vertex $v \in V_{G_x} \cap V_{H_x}$ in H_x, the same is true in G_x. The reverse direction is trivial. □

Given a vertex v of G_x that is not 4-connected to x, we still have to show how we can determine a 3-separator S that separates v and x. Once again, instead of searching for a 3-separator in G_x, we search for a 3-separator in H_x:

Lemma 10. *Let v be a vertex of G_x and let S be a 3-separator separating v and x in H_x. Then S is also a 3-separator separating v and x in G_x.*

Proof. Assume that the lemma above is not true. Let us consider the first replacement of $G_x = (V_{G_x}, E_{G_x})$ by a graph $G_x^* = (V_{G_x^*}, E_{G_x^*})$, and of $H_x = (V_{H_x}, E_{H_x})$ by a graph $H_x^* = (V_{H_x^*}, E_{H_x^*})$ such that the assertion of the lemma holds before, but not after, the replacement. It is clear that every 3-separator $T \subseteq V_{G_x^*}$ in H_x^* that separates a vertex $w \in V_{G_x^*}$ and x also separates w and x in G_x^*, since G_x^* is a subgraph of H_x^*. Thus, if the lemma does not hold for G_x^* and H_x^*, there must be a 3-separator T with $T \not\subseteq V_{G_x^*}$ that separates a vertex $w \in V_{G_x^*}$ and x in H_x^*. Since H_x^* is triconnected, there are three pairwise internally vertex-disjoint paths from x to w in H_x^*. Then, as shown in the proof of Lemma 9, there also exist three pairwise internally vertex-disjoint paths q_1, q_2, and q_3 from x to w in H_x^* that do not visit any vertex outside G_x^*. Hence, at least one vertex of T is not visited by q_1, q_2, and q_3. But this contradicts Lemma 3. □

For determining a 3-separator of H_x separating a vertex v and x, we can again use the data structure of Kanevsky et. al.. This data structure also maintains, in $O(\alpha(m, n))$ amortized time per edge insertion, a special decomposition tree from which, for any given pair of two non-4-connected vertices u and w, one can

construct, in constant time, two sets S_1 and S_2 such that one of these sets is a 3-separator separating u and w (see [9] for more details). Now, for two sets S_1 and S_2 with one of them being a 3-separator separating v and x, we start two interleaved depth-first searches on $G_x - S_1$ and on $G_x - S_2$ with v as the source node; and we continue until one of the two depth-first searches has completed its depth-first search tree containing vertex v without having visited vertex x. Depending on whether this happens to the depth-first search on $G_x - S_1$ or to that on $G_x - S_2$, either S_1 (in the first case) or S_2 (in the second case) is a 3-separator separating v and x. Since the running time of the subroutine above is dominated by that of the depth-first search that detects a 3-separator $S = S_1$ or $S = S_2$, and, after identifying S, all vertices and edges visited by this depth-first search will be deleted from G_x in order to complete the reduction step replacing G_x by the Δ-replacement G_x^* of G by S, the extra running time for determining 3-separators of two possible candidates can be bounded by $O(m)$, taken over all reduction steps.

Let us now analyze the complexity of the whole algorithm. For answering all 4-connectivity queries and identifying 3-separators separating a vertex v and x, we need only $O(m\alpha(m,n))$ time using the data structure of Kanevsky et. al., since this data structure can be initialized in $O(m\alpha(m,n))$ time and our algorithm consists of only $O(n)$ edge insertions and 4-connectivity queries (note that $n \leq m$, since G is triconnected). Since we have already shown that the remaining parts of our algorithm for identifying triangular cuts run in linear time, we can conclude that the following lemma holds:

Theorem 11. *Let $I = (G, s_1, s_2, t_1, t_2)$ be an instance of the 2-VDPP, where $G = (V, E)$ is an undirected triconnected graph. Then, in $O(m\alpha(m,n))$ time, an instance $I' = (G', s_1, s_2, t_1, t_2)$ of the 2-VDPP can be constructed such that I is 2P-reducible to I', and such that $G' = (V', E')$ is an undirected triconnected graph not containing any vertex $v \notin \{s_1, s_2, t_1, t_2\}$ that can be separated from $\{s_1, s_2, t_1, t_2\}$ by a triangular cut.*

With the results of Sect. 2 we can conclude:

Theorem 12. *The 2-VDPP can be solved in $O(n + m\alpha(m,n))$ time.*

5 Extensions

Perl and Shiloach [17] showed that the 2-EDPP on undirected graphs can be reduced to the 2-VDPP on undirected graphs as follows: If, for an instance (G, s_1, s_2, t_1, t_2) of the 2-EDPP, there are two vertex-disjoint paths from s_1 to t_1 and from s_2 to t_2, just output two such paths with an algorithm for the 2-VDPP. Otherwise, add a new vertex x and edges $\{x, s_1\}, \{x, s_2\}, \{x, t_1\}$ and $\{x, t_2\}$ to G. Then, there are two edge- (but not vertex-)disjoint paths from s_1 to t_1 and from s_2 to t_2, if and only if there is a vertex u of G that is 4-edge-connected to x. Given such a vertex u, one can use network-flow techniques to determine, in $O(m)$ time, four edge-disjoint paths from u to s_1, s_2, t_1, and t_2, and by concatenating two

of these paths it is easy to construct, in linear time, two edge-disjoint paths from s_1 to t_1 and s_2 to t_2. In [17] the problem of determining a vertex u that is 4-edge-connected to x was solved by n applications of network-flow techniques, which yields a running time of $O(mn)$. But, as shown by Dinitz and Westbrook [3], a sequence of q 4-edge-connectivity queries in an undirected graph with n vertices and m edges can be answered in $O(q + m + n \log n)$ total time. Hence, the 2-EDPP on undirected graphs can be solved in $O(m\alpha(m, n) + n \log n)$ time.

In [13] Lucchesi and Giglio presented a linear-time algorithm that, given an instance $I = (G, s_1, s_2, t_1, t_2)$ of the decision version of the 2-VDPP on dags with $G = (V, E)$, constructs an instance $I' = (G', s_1, s_2, t_1, t_2)$ of the decision version of the 2-VDPP for undirected graphs with $G' = (V', E')$ such that $|E'| \leq |E|$, $|V'| \leq |V|$, and there are two disjoint paths from s_1 to t_1 and from s_2 to t_2 in G iff the same is true of G'. Thus, there is an $O(m\alpha(m, n) + n)$-time algorithm for solving the decision version of the 2-VDPP on dags. Finally, similarly to the reduction of the 2-EDPP to the 2-VDPP on undirected graphs, for the 2-EDPP on a dag, either two disjoint paths can be found with an algorithm for the 2-VDPP, or we add two vertices x and y and directed edges $(x, s_1), (x, s_2), (t_1, y)$, and (t_2, y) to our input graph. In the latter case, there are two edge- (but not vertex-)disjoint paths, p_1 from s_1 to t_1 and p_2 from s_2 to t_2, iff there is a vertex u such that there are two edge-disjoint paths leading from x to u as well as two edge-disjoint paths from u to y. u, if it exists, can be determined in $O(n + m \log_{2+m/n} n)$ time with a data structure of Suurballe and Tarjan [25]. Hence, the decision version of the 2-EDPP on dags is solvable in $O(n + m \log_{2+m/n} n)$ time.

6 Acknowledgements

My special thanks go to Jens Gustedt for his contributions to our email discussion on the 2-disjoint paths problem. The feedback and friendly support I received from Jens Gustedt have been a constant encouragement for me to pursue my objective and solve the 2-VDPP in nearly linear time. I am indebted to Torben Hagerup for his advice and many helpful comments on preversions of this paper.

References

1. A. Aggarwal, J. Kleinberg, and D. P. Williamson, Node-disjoint paths on the mesh and a new trade-off in VLSI layout, *SIAM J. Comput.* **29** (2000), pp. 1321–1333.
2. J. Bang-Jensen and G. Gutin, *Digraphs: Theory, Algorithms and Applications*, Springer, London, 2001.
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. Even, A. Itai, and A. Shamir, On the complexity of timetable and multicommodity flow problems, *SIAM J. Comput.* **5** (1976), pp. 691–703.
5. S. Fortune, J. Hopcroft, and J. Wyllie, The directed subgraph homeomorphism problem, *Theoret. Comput. Sci.* **10** (1980), pp. 111–121.

6. C. P. Gopalakrishnan and C. Pandu Rangan, Edge-disjoint paths in permutation graphs, *Discuss. Math. Graph Theory* **15** (1995), pp. 59-72.
7. J. Gustedt, The general two-path problem in time $O(m \log n)$ (extended abstract), Report No. 394/1994, TU Berlin, FB Mathematik, 1994.
8. D. Jungnickel, *Graphen, Netzwerke und Algorithmen*, B. I. Wissenschaftsverlag, Mannheim, 1994.
9. A. Kanevsky, R. Tamassia, G. Di Battista, and J. Chen, On-line maintenance of the four-connected components of a graph, Proc. 32nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 1991), pp. 793–801.
10. R. M. Karp, On the computational complexity of combinatorial problems, *Networks* **5** (1975), pp. 45–68.
11. S. Khuller, S. G. Mitchell, and V. V. Vazirani, Processor efficient parallel algorithms for the two disjoint paths problem and for finding a Kuratowski homeomorph, *SIAM J. Comput.* **21** (1992), pp. 486–506.
12. E. Korach and A. Tal, General vertex disjoint paths in series-parallel graphs, *Discrete Appl. Math.* **41** (1993), pp. 147–164.
13. 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.
14. J. F. Lynch, The equivalence of theorem proving and the interconnection problem, *(ACM) SIGDA Newsletter,* **5** (1975), pp. 31–36.
15. 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.
16. 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.
17. Y. Perl and Y. Shiloach, Finding two disjoint paths between two pairs of vertices in a graph, *J. ACM* **25** (1978), pp. 1–9.
18. N. Robertson and P. D. Seymour, Graph minors. XIII. The disjoint paths problem, *J. Comb. Theory, Ser. B*, **63** (1995), pp. 65–110.
19. P. Scheffler, A practical linear time algorithm for disjoint paths in graphs with bounded tree-width, Report No. 396/1994, TU Berlin, FB Mathematik, 1994.
20. A. Schrijver, A group-theoretical approach to disjoint paths in directed graphs, *CWI Quarterly* **6** (1993), pp. 257–266.
21. A. Schrijver, *Combinatorial optimization – Polyhedra and Efficiency Vol. C*, Springer, Berlin, 2002.
22. A. Schwill, Nonblocking graphs: greedy algorithms to compute disjoint paths, Proc. 7th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1990), Lecture Notes in Computer Science Vol. 415, Springer-Verlag, Berlin, 1990, pp. 250-262.
23. P. D. Seymour, Disjoint paths in graphs, *Discrete Math.* **29** (1980), pp. 293–309.
24. Y. Shiloach, A polynomial solution to the undirected two paths problem, *J. ACM* **27** (1980), pp. 445–456.
25. J. W. Suurballe and R. E. Tarjan, A quick method for finding shortest pairs of disjoint paths. *Networks* **14** (1984), pp. 325-336.
26. C. Thomassen, 2-linked graphs, *Europ. J. Combinatorics* **1** (1980), pp. 371–378.
27. M. E. Watkins, On the existence of certain disjoint arcs in graphs, *Duke Math. J.* **35** (1968), pp. 231–246.
28. S. G. Williamson, Depth-first search and Kuratowski subgraphs, *J. ACM* **31** (1984), pp. 681–693.