Solving the 2-Disjoint Paths Problem in Nearly Linear Time

Torsten Tholey

2005

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 only $O(n+m\alpha(m,n))$ 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 2-vertex- and the 2-edge-disjoint paths problem on dags.

Copyright

The original publication appeared in Lecture Notes of Computer Science (LNCS) and is available at www.springerlink.com. The copyright is held by Springer.

Solving the 2-Disjoint Paths Problem in Nearly Linear Time

Torsten Tholey

Institut für Informatik
Universität Augsburg
D-86135 Augsburg, Germany
tholey@informatik.uni-augsburg.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 only $O(n + m\alpha(m, n))$ 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 2-vertex- and the 2-edge-disjoint paths problem on dags.

1 Introduction

The construction of disjoint paths in a given graph G is one of the fundamental problems in graph theory with many applications, for example in the context of routing problems, network reliability, and VLSI-design. In the following subsection we will consider different versions of disjoint paths problems.

1.1 Disjoint paths problems

One of the most intensively studied disjoint paths problems can be described as follows. Given 2k not necessarily pairwise distinct vertices $s_1, \ldots, s_k, t_1, \ldots t_k$ construct, if possible, k disjoint paths p_i ($1 \leq i \leq k$) from the vertices $s_1, \ldots, s_k$ to the vertices $t_1, \ldots t_k$ such that p_i leads from s_i to $t_{\sigma(i)}$ and $\sigma : \{1, \ldots, k\} \to \{1, \ldots, k\}$ defines an arbitrary permutation of the numbers $1, \ldots, k$ (if $s_1, \ldots, s_k, t_1, \ldots t_k$

are not pairwise distinct and if we search for vertex-disjoint paths, we should allow the endpoints of the paths to overlap). It is well known that this paths problem can be considered as a network flow problem and can be solved in linear time (cf. [9]).

In this paper we will focus on a more complicated paths problem, namely the *k-disjoint paths problem*. In this problem, given $2k$ vertices $s_1, \ldots, s_k, t_1, \ldots, t_k$, we want to construct k disjoint paths p_i from s_i to t_i $(1 \leq i \leq k)$. We will also refer to this problem as k-DPP or, more precisely, as k-VDPP if disjoint means vertex-disjoint, and as k-EDPP if disjoint means edge-disjoint. W.l.o.g. throughout this paper we will assume that the vertices $s_1, \ldots, s_k, t_1, \ldots, t_k$ are pairwise distinct. If necessary, this condition can be achieved by applying some simple reductions. We will give an overview of the known results concerning the k-DPP in Sect. 1.2. Further overviews can be found in [1], [25], and [26]. 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 $(1 \leq i \leq k)$. Fortunately, an algorithm for the decision version of the k-DPP can also be used for solving our version of the problem:

Lemma 1. *Given an $O(T(m,n))$-time algorithm for the decision version of the k-DPP, k disjoint paths solving the k-DPP, if they exist, can be computed within $O(n + kmT(m,n))$ time. The result holds for both versions of the problem, the k-EDPP and the k-VDPP.*

Interestingly, this lemma does not appear in the standard literature for the k-DPP (possibly, because of its simple proof). We therefore want to prove this lemma before starting with our overview of known results concerning the k-DPP.

Proof of Lemma 1. We start with constructing the path from s_k to t_k by adding step by step one edge to this paths such that the remaining sligthly modified disjoint paths problem remains feasible. The feasability is tested with the $O(T(m,n))$-time algorithm. Following this approach, after $O(mT(m,n))$ time the problem is reduced to the $(k-1)$-disjoint paths problem that can be solved recursively in the same way. □

1.2 Results for the k-DPP

Previous results on undirected graphs. Let us first consider the k-DPP on undirected graphs. If k is not fixed, as defined above, but is 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 for undirected graphs. This was shown by Knuth, cf. [11], and Lynch [16] 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 [18], Seymour [27], Shiloach [28], and Thomassen [31]. More precisely, Seymour gave a solution only for the decision version of the 2-DPP, but for both, the 2-EDPP and the 2-VDPP. Ohtsuki, Shiloach, and Thomassen only considered the 2-VDPP. However, with a reduction described by Perl and Shiloach [20] 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 [12] 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. 7 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] do not hold in general, 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 vertex-cut 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.

For the more general k-DPP, Robertson and Seymour [21] showed that the decision version of the undirected k-VDPP is solvable in $O(n^3)$ time. Finally, Perković and Reed [19] improved the running time to $O(n^2)$, which is currently the best known time bound for the decision version of the k-VDPP on undirected graphs, for all $k \geq 3$. Unfortunately, the constant hidden in the big-O-notation of the time bound of Robertson's and Seymour's algorithm being very large, Robertson's and Seymour's algorithm is only of theoretical

interest, even with the improvements of Perković and Reed. Until now, there is no algorithm of practical importance known to solve the k-VDPP for a $k \geq 3$.

Previous results on directed graphs. 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, Perl and Shiloach [20] 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 [15] described a linear time reduction from the decision version of the 2-VDPP on dags to the 2-VDPP on undirected graphs. From a corollary of their paper it also follows that there are always two paths solving the 2-VDPP on the undirected graph after the reduction if this graph is non-planar; and since Perl and Shiloach [20] 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.[1] As the author of this paper showed in [33], the reduction of Lucchesi and Giglio can be modified so as to reduce the general 2-VDPP or 2-EDPP on dags in $O(m \log_{2+m/n} n + n \log^3 n)$ time to the 2-VDPP on undirected graphs. As an application of the k-EDPP Schrijver [23] described an airplane routing problem that can be solved with an algorithm for the k-EDPP on dags.

Further results. For many special undirected or directed graphs, the 2-DPP or the more general k-DPP is solvable in linear time (see [20], [6], [13], and [22]).

New results. In this paper we present an algorithm for solving the 2-VDPP on undirected graphs, which improves the previously known best time bound of $O(n^2)$ to $O(n + m\alpha(m, n))$. Here, α denotes the inverse of the Ackermann function as defined by Tarjan and van Leeuwen [30]. Applying a slightly modified version of a well-known reduction, we further show that our result leads to an $O(m\alpha(m, n) + n \log n)$ time bound for the 2-EDPP on undirected graphs. With the $O(m \log_{2+m/n} n + n \log^3 n)$-time reduction in [33] our new algoritm for the 2-VDPP can also be used to solve the 2-VDPP and 2-EDPP on dags in $O(m \log_{2+m/n} n + n \log^3 n)$ time.

[1] This was not recognized by the author in the conference version [32] of this paper.

The main result of this paper being a new algorithm for the 2-VDPP on undirected graphs, Sect. 2 up to Sect. 6 will be concerned with undirected graphs only. If, in Sect. 2 up to Sect. 6, we refer to disjoint paths we always mean vertex-disjoint paths.

2 The main ideas

In this section we will introduce the main ideas underlying the algorithm presented in this paper for solving the 2-VDPP on undirected graphs. A detailed description will be given in the following sections.

We start with a few formal definitions. A path p in a graph G is called simple if no vertex appears on p more than once. For simplicity, in the rest of the paper, if we refer to paths, we always mean simple paths without mentioning it explicitely. Note that, if there are two non-simple disjoint paths p_i from s_i to t_i $(1 \leq i \leq 2)$, there also two simple disjoint paths p_i from s_i to t_i. Throughout this paper, an instance of the 2-VDPP is given by a tupel (G, s_1, s_2, t_1, t_2), where $G = (V, E)$ is a graph and s_1, s_2, t_1, and t_2 are the vertices of G for which we want to find two disjoint paths leading from s_i to t_i for $1 \leq i \leq 2$. 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. *Solving an instance* I *of the* 2-VDPP means determining whether there is a solution to I and, if so, to determine two paths p_1 and p_2 that solve I. *Solving the* 2-VDPP means implementing an algorithm able to solve every instance of the 2-VDPP.

Solving the 2-VDPP on general undirected graphs G seems to be complicated. Thus, we will reduce the problem to simpler graphs. More precisely, we will increase the connectivity of G step by step (for a formal definition of the connectivity k of a graph see Sect. 3).

Intuitively, as we increase the connectivity of G we also increase the quotient of the number m of edges divided by the square of the number n of vertices of G, and as a result G becomes a "denser" graph. If G is dense enough, we can always find two disjoint paths solving the 2-VDPP on G. For example, if G is a $(n-1)$-connected graph, then G is the complete graph with edges between every pair of vertices, and, hence, in the complete graph we always can find two disjoint paths between two given pairs of vertices. However, we hope

there is also a $k \in O(1)$ such that, for every k-connected graph, all instances of the 2-VDPP have a solution.

Basically, our algorithm for solving the 2-VDPP follows the approach of Shiloach [28] that can be described as follows: Following the idea of increasing the connectivity of G, we show in Sect. 4 that an instance of the 2-VDPP can be reduced, in linear time, to an other instance (G, s_1, s_2, t_1, t_2) of the 2-VDPP with G being a triconnected graph. The reduction follows ideas of Itai, which are sketched by Shiloach in [28]. In this paper we will describe in full detail how these ideas can be implemented with the help of block-cutpoint graphs. However, unlike Itai, with a simple trick we can avoid the computation of triconnected components. Unfortunately, a connectivity of $k = 3$ will not be sufficient to guarantee a solution for the 2-VDPP as we will see at the end of Sect. 4. However, there is a very simple linear time algorithm of Woeginger [37] for the 2-VDPP on planar triconnected graphs and, as observed by Shiloach [28], the 2-VDPP always has a solution in the case of a 4-connected nonplanar graph. In Sect. 5 we describe the main ideas of solving the 2-VDPP on 4-connected graphs. Since a 4-connected graph is a triconnected graph without any 3-vertex-cut, in order to reduce an instance of the 2-VDPP on a triconnected graph G to an instance on a 4-connected graph, we should try to remove all 3-vertex-cuts from G. A more detailed look at the results given in Sect. 5 will show that in order to guarantee a solution for the 2-VDPP we do not need to delete all 3-vertex-cuts from G. However, all 3-vertex-cuts that separate a vertex v from $\{s_1, s_2, t_1, t_2\}$ should be deleted. In Sect. 6 we give a short sketch of Shiloach's approach for deleting such cuts leading to a running time of $O(mn)$ for the 2-VDPP. We then present a more efficient algorithm to delete the 3-vertex-cuts. This new algorithm leads to a running time of $O(n + m\alpha(m, n))$ for solving the 2-VDPP, which is the main result of this paper. In Sect. 7 we show how the new algorithm for the 2-VDPP can be used to improve the running time of the 2-EDPP.

3 Facts and definitions

Definition 2. *Two vertices v and w are k-(vertex-)connected iff $v = w$ or there are k internally vertex-disjoint[2] paths from v to w. We also say that v is k-connected to w. An undirected graph G is called k-connected iff every pair of vertices v and w in G is k-connected. We further say that, in this case, G has connectivity k.*

An alternative characterization of k-connected graphs uses k-vertex-cuts:

Definition 3. *Let $k \in \mathbb{N}$ and let $G = (V, E)$ be an undirected graph with $n \geq k + 2$ vertices. Then, a (k-)vertex-cut (of G) is a set C of k vertices such that $G - C$ is not connected.[3] If two vertices v and w of $G - C$ are not connected in $G - C$ we say that C separates v and w (in G) or that v and w are separated by C. We also say that C separates a vertex $v \in V - C$ (in G) from a set $S \subseteq V$ or that v can be separated from S by C (in G), if the connected component of $G - C$ containing v does not contain any vertex of S.*

Note that for a vertex-cut C, we have defined "C separates two vertices v and w" as it is usual in graph theory (cf. [24]), whereas, for technical reasons, we have defined "C separates a vertex v from a vertex set S" in an unusual way. For two vertices v and w, "C separates v and w", "C separates v from $\{w\}$", and "C separates w from $\{v\}$" are three different assertions.

Theorem 4 (Menger's theorem, [17][4]). *Let $k \in \mathbb{N}$, and let $G = (V, E)$ be an undirected graph with $n \geq k + 2$ vertices. Then G is k-connected if there exists no $(k-1)$-vertex-cut of G. Two vertices $v, w \in V$ with $(v, w) \notin E$ are k-(vertex)-connected, if there is no $(k-1)$-vertex-cut of G separating v and w. Two vertices $v, w \in V$ with $(v, w) \in E$ are k-(vertex)-connected, if there is no $(k-2)$-vertex-cut of $G - \{(v, w)\}$ separating v and w.*

[2] By internally vertex-disjoint we mean that each pair of the disjoint paths apart from their endpoints has no vertex in common.

[3] 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 $(V, E - F)$.

[4] More precisely, Menger considered pairs of non-adjacent vertices. The results concerning adjacent vertices were proven by Whitney [35].

In a k-connected graph it is possible to construct disjoint paths from one vertex to k different vertices:

Lemma 5. *Given $k+1$ pairwise distinct vertices $v_0, v_1, v_2, \ldots, v_k$ in a k-connected graph G, there are k paths p_i from v_0 to v_i for $i \in \{1, 2 \ldots, k\}$ such that the paths have no vertex in common, except vertex v_0.*

A proof of this lemma can be found in [9] (Exercise 8.1.3).

Corollary 6. *Given k pairwise distinct vertices $v_1, v_2, \ldots, v_k$ as well as k pairwise distinct vertices $w_1, w_2, \ldots, w_k$ in a k-connected graph G, there exist k pairwise disjoint paths p_i for $i \in \{1, 2, \ldots, k\}$ such that p_i leads from v_i to a vertex in $\{w_1, w_2, \ldots, w_k\}$.*

A proof can be found in [9] (Exercise 7.1.7 in combination with Exercise 7.1.9). Another important and obvious fact about k-vertex-cuts will be used in the later part of this paper, namely:

Corollary 7. *Let $p_1, p_2, \ldots, p_k$ be a set of k internally disjoint paths in an undirected graph $G = (V, E)$ leading from a vertex $v \in V$ to a vertex w. Then, if there is a k-vertex-cut C separating v and w, every path p_i ($1 \leq i \leq k$) visits exactly one vertex of C.*

In the following we use the word "biconnected" as synonym for "2-connected" and the word "triconnected" as synonym for "3-connected". For a 1-vertex-cut C the vertex v with $v \in C$ will be referred to as a *cutpoint*. Queries, referred to as k-connectivity queries, ask whether two given vertices v and w are k-connected. A *block* of a connected graph $G = (V, E)$ is a maximal subgraph $D = (V_D, E_D)$ of G with $V_D \subseteq V$ and $E_D = E \cap V_D \times V_D$ without any cutpoint. By maximal we mean that there exists no other subgraph $F = (V_F, E_F)$ of G with $E_F = E \cap V_F \times V_F$ and without any cutpoint, for which $V_D \subset V_F \subseteq V$ holds. For example, the graph in Fig. 1.a has a block consisting of the vertices d, e, f, and g, and edges $(d, e), (d, f), (d, g), (e, f)$, and (f, g). One important property of blocks is given in the following lemma (cf. [9], Sect. 8):

Lemma 8. *For two different blocks $B_1 = (V_1, E_1)$ and $B_2 = (V_2, E_2)$ we always have $|V_1 \cap V_2| \leq 1$. If there is a vertex $v \in V_1 \cap V_2$, then v is a cutpoint.*

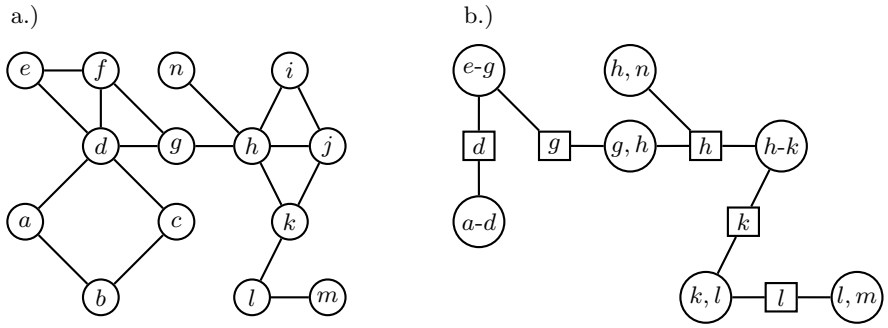

Fig. 1. a.) A graph and b.) its block-cutpoint graph.

The *block-cutpoint graph* $B(G)$ of a given connected graph G is the graph having the set of cutpoints and blocks of G as its node set and all pairs (c, B), where c is a cutpoint of block B, as its arcs. Fig. 1.b shows the block-cutpoint graph of the graph in Fig. 1.a. Blocks are denoted by circles, cutpoints by squares. For a better understanding we refer to "nodes" and "arcs" if we mean the nodes and arcs of the block-cutpoint graph, and to "vertices" and "edges" if we mean the vertices and edges of G. It is well-known that the block-cutpoint graph of an undirected connected graph is always a tree that can be constructed in linear time (cf. [9]). For a block-cutpoint graph $B(G)$ of a connected graph $G = (V, E)$, we let $b_{B(G)}(w)$ be equal to node w if vertex w is a cutpoint, and, if it is not, $b_{B(G)}(w)$ be equal to the block containing w. We usually omit the index $B(G)$ if it is clear from the context. In this paper, if $b(w)$ is equal to a block we call it a *block node*, and, if it is not, a *cutpoint node*.

The following lemma concerning block-cutpoint graphs will be useful for the following sections.

Lemma 9. *For a pair of vertices v and w of an undirected connected graph G, let p be the (tree) path in the block-cutpoint graph $B(G)$ from $b(v)$ to $b(w)$. Then all cutpoint nodes that are visited by p must also be visited as vertices by every path from v to w in G, and, beside v and w, the cutpoint nodes visited by p are the only vertices that must be visited by every path from v to w in G.*

4 Reducing the 2-VDPP to triconnected graphs

In this section we want to reduce the 2-VDPP on general graphs in several steps to the 2-VDPP on triconnected graphs. Shiloach mentioned in [28] that there is a linear time algorithm of Itai for such a reduction beginning with the following steps:

1. Reduce the 2-VDPP on an instance $I = (G, s_1, t_1, s_2, t_2)$ of the 2-VDPP to an instance $I' = (G', s'_1, t'_1, s'_2, t'_2)$ such that G' is biconnected.
2. Reduce the 2-VDPP on the instance (G, s_1, t_1, s_2, t_2) resulting from step 1 to an instance $(G', s'_1, t'_1, s'_2, t'_2)$ such that G' is biconnected and there is no 2-vertex-cut S of G' containing one of the vertices s'_1, t'_1, s'_2, or t'_2.
3. Let $I = (G, s_1, t_1, s_2, t_2)$ be the instance of 2-VDPP resulting from the first two steps. For each 2-vertex-cut $\{u, v\}$ such that there is a connected component D of $G - \{u, v\}$ containing none of the vertices s_1, s_2, t_1, and t_2, remove all vertices of D with their adjacent edges from G and insert an edge between u and v, if such an edge does not already exist.

Unfortunately, no explicit implementation of these steps are given in Shiloach's paper. Hence, in this paper we will give an explicit implementation of the above steps using block-cutpoint graphs. Our implementation is chosen in such a way that after the above three steps we will obtain an instance $I = (G, s_1, s_2, t_1, t_2)$ of the 2-VDPP with G being a triconnected graph, whereas Itai's original algorithm consists of some further steps. Hence, our algorithm is much simpler than Itai's algorithm and avoids the computation of the triconnected components of a graph.

For our different reduction steps we use a special kind of reduction algorithm that we will call *2PR-algorithm*.

4.1 2PR-algorithms

Definition 10. *We call an algorithm a 2-paths reduction algorithm (or, for short, a 2PR-algorithm) if, started on an instance $I_1 = (G, s_1, s_2, t_1, t_2)$ of the 2-VDPP with $G = (V, E)$, it either solves I_1 or determines an instance $I_2 = (G', s'_1, s'_2, t'_1, t'_2)$ of the 2-VDPP with $G' = (V', E')$ such that the following properties hold:*

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

Then, we also say that I_1 is 2P-reduced to I_2.

As an example, in a first step, we want to reduce the 2-VDPP to connected graphs:

It is easy to see that for an instance $I = (G, s_1, s_2, t_1, t_2)$ of the 2-VDPP disjoint paths p_1 from s_1 to t_1 and p_2 from s_2 to t_2 can only exist, if for $i \in \{1, 2\}$, the vertices s_i and t_i are part of the same connected component of G. If this is the case, but s_1 and s_2 are not contained in the same connected component, a solution to the 2-VDPP must exist and can be computed in linear time using depth-first searches in the connected components of s_1 and s_2 to construct paths from s_i to t_i for $i \in \{1, 2\}$. If s_1, s_2, t_1, t_2 are all part of the same connected component C of G we can replace G by C. Every solution on (C, s_1, s_2, t_1, t_2) then also is a solution to I and it is easy to see that all properties of Definition 10 hold, if we define $I_1 = I$ and $I_2 = (C, s_1, s_2, t_1, t_2)$. Hence, we have just presented a 2PR-algorithm reducing the 2-VDPP on general graphs to the 2-VDPP on connected graphs.

4.2 2P-Reduction of the 2-VDPP to biconnected graphs

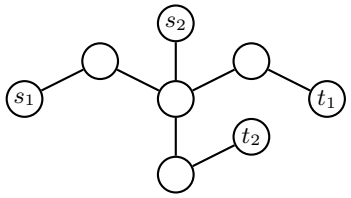

Fig. 2. A connected graph without a solution for the 2-VDPP.

Fig. 2 shows that instances $I = (G, s_1, s_2, t_1, t_2)$ of the 2-VDPP, where G is a connected graph, do not always have a solution. Hence,

in this section, we describe a 2PR-algorithm that, given an instance $I = (G, s_1, s_2, t_1, t_2)$, where G is a connected graph, either solves I or 2P-reduces I to an instance $I' = (G', s'_1, s'_2, t'_1, t'_2)$ with G' being a biconnected graph. This corresponds to Step 1 of Itai's algorithm.

Given an instance $I = (G, s_1, s_2, t_1, t_2)$ of 2-VDPP, where G is a connected graph, we first construct the block-cutpoint graph $B(G)$ of G. We root the block-cutpoint tree in node $b(s_1)$ (remember the definition of $b(s_1) = b_{B(G)}(s_1)$ in Sect. 3), and, for $i \in \{1, 2\}$, we define q_i to be the tree path from $b(s_i)$ to $b(t_i)$ in $B(G)$.

Let v be the lowest common ancestor of $b(s_2)$ and $b(t_2)$ in $B(G)$. We consider the following cases.

Case 1: v is not visited by q_1. Then we can construct two disjoint paths p_1 from s_1 to t_1 and p_2 from s_2 to t_2 in G as follows: We construct p_1 from q_1 by replacing each sub-path of q_1 consisting of three consecutive nodes c_1, b, c_2 such that c_1 and c_2 are cutpoint nodes, and b is a block node, by a path from c_1 to c_2 in G only visiting vertices of the block represented by b, but not visiting node v, if v is a cutpoint node. Note that such a path from c_1 to c_2 must exist, since block b is biconnected. Similarly, if $b(s_1)$ is a block node and c is a cutpoint node visited immediately after vertex $b(s_1)$ by q_1 we replace the sub-path of q_1 from $b(s_1)$ to c by a path from s_1 to c in G that only visits vertices $w \in b(s_1)$ and, if v is a cutpoint node, does not visit vertex v; if $b(t_1)$ is a block node and c is a cutpoint node visited immediately before $b(t_1)$ by q_1, we replace the sub-path of q_1 leading from c to $b(t_1)$ by a path from c to t_1 in G that only visits vertices $w \in b(t_1)$ and, if v is a cutpoint node, does not visit vertex v. In the special case where $b(s_1) = b(t_1)$ we just choose for p_1 a path from s_1 to t_1 in $b(s_1)$ that, if v is a cutpoint node, does not visit v. p_2 is constructed in the same way but instead of not visiting v we do not allow p_2 to visit the father of v in $B(G)$ if the father of v is a cutpoint node.

To see that p_1 and p_2 are vertex-disjoint, we can conclude from Lemma 8 that the paths can only intersect each other in one or more cutpoints. But, for $1 \leq i \leq 2$, the set C_i of all cutpoints visited by path p_i is a subset of the set of all cutpoint nodes visited by q_i or neighbored to a block node visited by q_i. Thus, $C_1 \cap C_2$ can contain at most one vertex either equal to v, if v is a cutpoint node, or equal to the father of v, if the father of v is a cutpoint node. However, we

have already shown that in the first case p_1 does not visit vertex v, and in the second case p_2 does not visit the father of v. Hence, p_1 and p_2 are disjoint.

Case 2: v is visited by q_1. Then, if v is a cutpoint node, according to Lemma 9 v must be visited by every path from s_1 to t_1 in G as well as by every path from s_2 to t_2 in G. Hence, in this case, our instance of the 2-VDPP has no solution. The same is true if one of the children of v in $b(G)$ is visited by both paths, q_1 and q_2 (then v or its child is a cutpoint node and this cutpoint must be visited by every path p_i from s_i to t_i $(1 \leq i \leq 2)$).

Let us now assume that v is not a cutpoint node and that there is no child of v in $B(G)$ visited by q_1 and q_2. For $i \in \{1, 2\}$, let us define s'_i to be the cutpoint node visited by q_i immediately before v, or, if such a cutpoint node does not exist, $s'_i := s_i$, and, similarly, let us define t'_i to be the cutpoint node visited by q_i immediately after v, or, if such a cutpoint node does not exist, $t'_i := t_i$. Then, as in Case 1, it is easy to construct, for $i \in \{1, 2\}$, disjoint paths from s_i to s'_i and t'_i to t_i such that every path visits only vertices w with $b(w)$ lying on the sub-path of q_i from $b(s_i)$ to $b(s'_i)$ or from $b(t'_i)$ to $b(t_i)$, respectively, and, moreover, s'_1, s'_2, t'_1, and t'_2 are the only vertices of block v that are visited by the constructed paths. Then, if there are two disjoint paths p'_1 from s'_1 to t'_1 and p'_2 from s'_2 to t'_2 the concatenation of the path from s_1 to s'_1, p'_1 and the path from t'_1 to t_1 and the concatenation of the path from s_2 to s'_2, p'_2, and the path from t'_2 to t_2 describe two disjoint paths that solve our instance of the 2-VDPP. Conversely, since, for $i \in \{1, 2\}$, the vertices s'_i and t'_i must be visited by every path from s_i to t_i, there can be no solution to the 2-VDPP in G if there are no two disjoint paths from s'_1 to t'_1 and s'_2 to t'_2 in block v. Hence, our problem can be reduced to the problem of finding two disjoint paths from s'_1 to t'_1 and s'_2 to t'_2 in the biconnected graph consisting of block v.

4.3 2P-Reduction of the 2-VDPP to triconnected graphs

Fig. 3 shows that instances $I = (G, s_1, s_2, t_1, t_2)$ of the 2-VDPP, where G is a biconnected graph, do not always have a solution. Hence, in this section, we describe a 2PR-algorithm that, given an instance $I = (G, s_1, s_2, t_1, t_2)$, where G is a biconnected graph, either

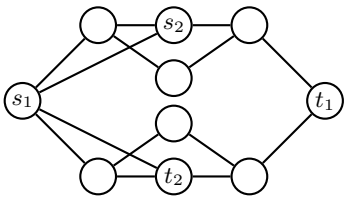

Fig. 3. A biconnected graph without a solution for the 2-VDPP.

solves I or 2P-reduces I to an instance $I' = (G', s'_1, s'_2, t'_1, t'_2)$ with G' being a triconnected graph. This reduction corresponds to the Steps 2 and 3 of Itai's algorithm.

In simple terms, our 2PR-algorithm in Sect. 4.2 consists of removing all cutpoints of our instance. We now want to remove all 2-vertex-cuts. We start with Step 2 of Itai's algorithm, but we split this step into two substeps.

2a: Remove all 2-vertex-cuts of the form $\{s_i, v\}$ or $\{t_i, w\}$ with $v, w \in V \setminus \{s_j, t_j\}$ and $\{i, j\} = \{1, 2\}$ that separates s_j and t_j.
2b: Remove all remaining 2-vertex-cuts C with $C \cap \{s_1, s_2, t_1, t_2\} \neq \emptyset$.

Step 2a: We can simply determine whether $\{s_i, t_i\}$ is a 2-vertex-cut separating s_j and t_j by simply testing if s_j and t_j are part of different connected components of $G - \{s_i, t_i\}$ and, if so, no solution of the 2-VDPP can exist. All other separators $\{s_i, v\}$ or $\{t_i, v\}$ with $v \in V - \{s_i, t_i, s_j, t_j\}$ and separating s_j and t_j can be removed by adding edges $(s_1, s_2), (s_1, t_2), (t_1, s_2)$, and (t_1, t_2) to G, if they do not already exist. This does not change the solvability of the 2-VDPP, because these edges cannot be used by two disjoint paths p_1 from s_1 to t_1 and p_2 from s_2 to t_2. However, for i, j with $\{i, j\} = \{1, 2\}$, in the new graph G there is no 2-vertex-cut $\{s_i, v\}$ or $\{t_i, v\}$ with $v \notin \{s_i, t_i, s_j, t_j\}$ separating s_j and t_j, since due to the edges (s_j, t_i) and (t_i, t_j) or (s_j, s_i) and (s_i, t_j), respectively, s_j and t_j are part of the same connected component in $G - \{s_i, v\}$ or $G - \{t_i, v\}$.

Step 2b: In order to remove all of the remaining 2-vertex-cuts $\{v, w\}$ with $s_1 \in \{v, w\}$, we construct the block-cutpoint graph of $G - \{s_1\}$. If the block-cutpoint graph consists of only one block node

and no cutpoint nodes there is no 2-vertex-cut C of G with $s_1 \in C$. Otherwise the set $\{(s_1, v) \mid v \text{ is a cutpoint of } G - \{s_1\}\}$ is exactly the set of all 2-vertex-cuts of G containing vertex s_1. Let us now consider an arbitrary cutpoint node v of the block-cutpoint graph $B(G-\{s_1\})$. If we remove v from $B(G-\{s_1\})$ the resulting graph is divided in several connected components that are subtrees of $B(G-\{s_1\})$. More precisely, we have one subtree that we call the *parent tree of* v that contains the parent of v, if v has a parent, and several trees that contain exactly one child of v that we call *childtrees*. In the following we distinguish between two cases depending on whether $t_1 = v$ or not. We do not consider any case where there is a 2-vertex-cut $\{s_i, v\}$ or $\{t_i, w\}$ with $v, w \in V \setminus \{s_j, t_j\}$ and $\{i, j\} = \{1, 2\}$ that separates s_j and t_j, since we have shown that such cases can be excluded. We illustrate the following Cases 1 and 2 in Fig. 4, where, for simplicity, we let v be the root of the block-cutpoint graph $B(G - \{s_1\})$.

Case 1: $t_1 = v$. We know that $b(s_2)$ and $b(t_2)$ lie in the same parent or child tree T (otherwise $b(s_2)$ and $b(t_2)$ are separated by a 2-vertex-cut of G containing s_1). Since v is a cutpoint node, there is also another parent or child tree T' not containing $b(s_2)$. G being a biconnected graph, for every vertex w with $b(w) \in T'$, there must be two disjoint paths from w to t_2 one visiting vertex v and the other visiting vertex s_1. Hence, there is a vertex u with $b(u) \in T'$ and $(u, s_1) \in E$. We can conclude that in Case 1 we can construct two disjoint paths q_1 from $b(u)$ to $b(t_1)$ and q_2 from $b(s_2)$ to $b(t_2)$ in $B(G - \{s_1\})$. Using the same techniques as in Sect. 4.2, we can replace the paths q_1 and q_2 by two disjoint paths p'_1 from u to t_1 and p_2 from s_2 to t_2 in $G-\{s_1\}$. By concatenating edge (s_1, u) and p'_1, we now have constructed two disjoint paths thus solving our instance of the 2-VDPP (see Fig. 4).

Case 2: $t_1 \neq v$. Then, since t_1, s_2, and t_2 are connected by the edges $(s_2, t_1), (t_2, t_1)$, the child or parent tree T containing $b(t_1)$ contains at least one of the nodes $b(t_2)$ and $b(s_2)$ (note that if T contains exactly one of the nodes $b(t_2)$ and $b(s_2)$ the other one must be equal to v). As in Case 1 there must be another parent or child tree T' different from T that contains a node $b(u)$ for a vertex $u \in V$ with $(s_1, u) \in E$. Hence, there is a path from node v to node $b(u)$ in $B(G - \{s_1\})$ not visiting any node in T. This path can be replaced

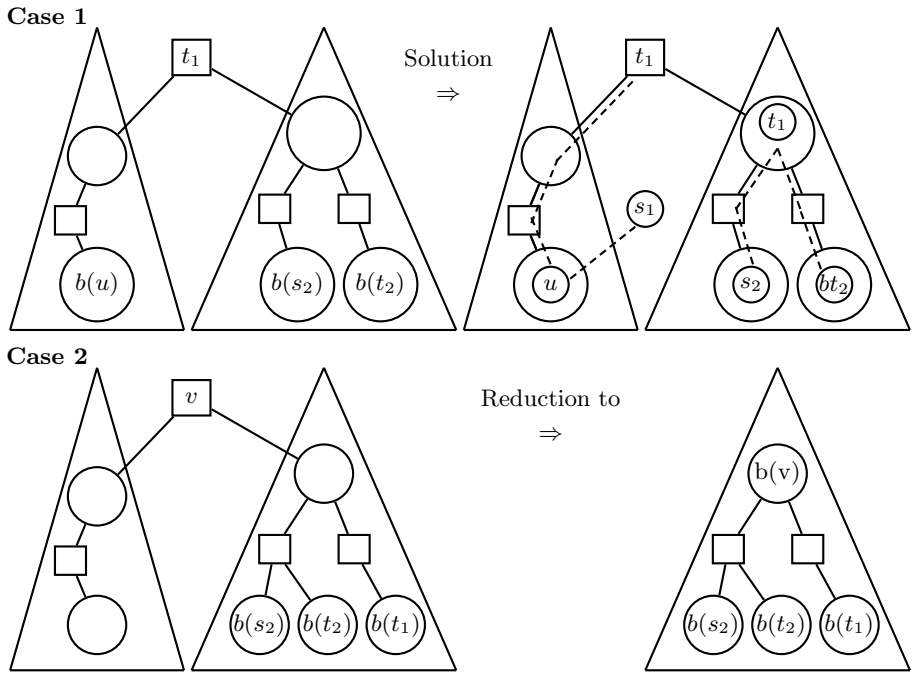

Fig. 4. Removing all 2-vertex-cuts $\{s_1, v\}$ with $v \in V - \{s_1\}$.

by a path p from v to u in G not visiting any vertex w with $b(w) \in T$. This means that, if there are disjoint paths p'_1 from v to t_1 and p_2 from s_2 to t_2 in G visiting only vertices w with $w = v$ or $b(w)$ in T, then p'_1 can be extended to a path from s_1 to t_1 such that p_1 and p_2 solve our original instance of the 2-VDPP. Conversely, it is easy to show that for two disjoint paths p_1 and p_2 solving our original instance of the 2-VDPP, only the path from s_1 to t_1 can visit vertex v and, if so, this path after visiting vertex v visits only vertices w with $b(w)$ in T. Hence, our instance of the 2-VDPP can be 2P-reduced to an instance (G', s_1, s_2, t_1, t_2), where G' is the graph obtained from G by deleting all vertices $w \in V - \{s_1, v\}$ for which $b(w)$ is not contained in T and by adding an edge (s_1, v) to G, if this edge does not already exist (note that this reduction also works in the special case where $v \in \{s_2, t_2\}$). If T consists of more than one block node we

recursively have to consider the block-cutpoint graph $B(G' - \{s_1\})$ which is equal to T.

We just have demonstrated how we can 2P-reduce our given instance $I = (G, s_1, s_2, t_1, t_2)$ of the 2-VDPP to a new instance $I' = (G', s_1, s_2, t_1, t_2)$ such that in G' there is no 2-vertex-cut C containing vertex s_1. Such a reduction does not create any new 2-vertex-cuts C that do not exist in G. It is easy to see that the reduction runs in linear time, since in Case 1 a solution can be constructed in $O(m)$ time, whereas in Case 2 we obtain a smaller graph in a time that is linear in the number of edges and vertices that are removed from G by the reduction. In the same way as we have removed all 2-vertex-cuts C containing vertex s_1 we can also remove all 2-vertex-cuts containing one of the vertices s_2, t_1, or t_2.

Step 3: In Step 3 we want to remove from G all 2-vertex-cuts C for which there is a connected component D in $G - C$ with $D \cap \{s_1, s_2, t_1, t_2\} = \emptyset$. More precisely, given such a vertex-cut C and connected component D with $D \cap \{s_1, s_2, t_1, t_2\} = \emptyset$, we do not remove C from G but rather the connected component:

Lemma 11 (Itai, cf. Shiloach [28]). *Let C be a 2-vertex-cut of G, let D be a connected component of $G - C$ with $D \cap \{s_1, s_2, t_1, t_2\} = \emptyset$, and, finally, let G' be the graph obtained from G by removing all vertices of D with their adjacent edges and adding an edge between the vertices of C if this edge does not already exist. Then (G, s_1, s_2, t_1, t_2) is 2P-reduced to (G', s_1, s_2, t_1, t_2). Moreover, G' is biconnected.*

Since there is no explicit proof of this lemma in [28], we prove it explicitly in this paper:

Proof. Suppose we are given disjoint paths p_1 and p_2 in G with p_1 leading from a vertex v_1 of G' to a vertex w_1 of G' and p_2 leading from a vertex v_2 of G' to a vertex w_2 of G'. Since C is a 2-vertex-cut only one of the two paths can visit a vertex w in the connected component D of $G - C$ with $D \cap \{s_1, s_2, t_1, t_2\} = \emptyset$ and this path must visit one of the vertices of C before w and the other vertex of C after w. Hence, we can replace the sub-path between the two vertices of C in G by the edge between the two vertices of C in order to obtain disjoint paths from v_1 to w_1 and from v_2 to w_2 in G'. This

shows that there are two disjoint paths p_1 from s_1 to t_1 and p_2 from s_2 to t_2 in G' if there are such paths in G and that G' is biconnected.

Conversely, given disjoint paths p_1 from s_1 to t_1 and p_2 from s_2 to t_2 in G' such that one of the paths uses the edge between the vertices of C, we can construct disjoint paths from s_1 to t_1 and from s_2 to t_2 in G in the following way: We replace the edge between the vertices in C by a path between the vertices in C that apart from its endpoints only visits vertices of D. Since D is a connected component such a path must exist and can be found by a breadth-first search. It is obvious that given p_1 and p_2 the new paths can be constructed in a time linear in the number of edges and vertices of D, p_1, and p_2. □

After having reduced G to a smaller graph G', as defined in Lemma 11, we recursively apply Lemma 11 to G' and to all subsequently resulting graphs. Since there may be more than a constant number of graph replacements, we have to ensure that even a series of reductions reducing our original graph G to a new graph G^* results in a 2PR-algorithm, i.e., given two disjoint paths p_i between the vertices s_i and t_i for $1 \leq i \leq 2$ on G^* we can construct two disjoint paths solving our instance of the 2-VDPP on G in linear time. This is easy, since we only have to walk along the paths p_1 and p_2 and if we detect an edge e that was inserted by a reduction as described in Lemma 11, we replace it recursively as according to the proof of Lemma 11. Since we replace e by a path visiting only edges that are deleted by the reduction inserting edge e, and since each reduction step inserts only one edge, and, finally, since the edges deleted from G in different reductions are disjoint, the time needed for all edge replacements sums up to $O(m)$.

It remains to show how we can find a 2-vertex-cut C such that there is a connected component D of $G-C$ with $D \cap \{s_1, s_2, t_1, t_2\} = \emptyset$, and how we can find all components D with this property. We therefore insert an extra vertex x and edges $(x, s_1), (x, s_2), (x, t_1)$, and (x, t_2) to G. If we let G_x be the resulting graph, the following lemma holds:

Lemma 12. *Let $k \in \mathbb{N}$. For a vertex v in a graph G, there is a k-vertex-cut C such that v is contained in a connected component D*

of $G - C$ with $D \cap \{s_1, s_2, t_1, t_2\} = \emptyset$ if and only if x is not $(k+1)$-connected to v in G_x.

Proof. If, for a k-vertex-cut C, there is a connected component D of $G - C$ with $D \cap \{s_1, s_2, t_1, t_2\} = \emptyset$, then D is also a connected component of $G_x - C$, since adding an extra vertex x and edges between x and the vertices of $\{s_1, s_2, t_1, t_2\}$ will not produce any changes in D. Hence, if v is a vertex contained in D, x and v are not contained in the same connected component of $G_x - C$. This means that x is not $(k+1)$-connected to v in G_x.

Conversely, if x is not $(k+1)$-connected to v in G_x, then there is a k-vertex-cut C separating x and v in G_x. Since x is connected to the vertices s_1, s_2, t_1, and t_2 by the edges $(x, s_1), (x, s_2), (x, t_1)$, and (x, t_2), the vertices s_1, s_2, t_1, and t_2 lie either in C or in the same connected component as vertex x. Hence, none of the vertices s_1, s_2, t_1, t_2 lie in the same connected component as v in $G_x - C$, and we can conclude that the same holds for graph $G - C$. □

According to Lemma 11, in order to find a vertex that is separated from $\{s_1, s_2, t_1, t_2\}$ by a 2-vertex-cut C of G, we only need to search for a vertex in G_x that is not 3-connected to x. We can construct a list of such vertices by using a data structure of Di Battista and Tammassia [2] which after a preprocessing time of $O(m+n)$ allows us to test in constant time whether two vertices are 3-connected or not. Thus, if we want to know the vertices that are not 3-connected to x, a complete list of such vertices can be constructed in linear time.

In order to delete all vertices that are not 3-vertex-connected to x, we follow an aproach of an anonymous referee of this paper, since his aproach is shorter and easier than the original reduction used in a preversion of this paper:

Let us mark all vertices that are not 3-vertex-connected to x and let us define U to be the set of unmarked vertices. Then the following Lemma holds:

Lemma 13. *For each connected component D of $G_x - U$ the number of unmarked vertices v in G such that there is an edge between v and a vertex of D is exactly two.*

Proof. Since G_x is biconnected, there must be at least two unmarked vertices u_1 and u_2 with an edge leading from u_1 or u_2, respectively, to a vertex of D. Assume now that there are three unmarked vertices u_1, u_2, and u_3 that are connected to a vertex of D by an edge and let us define S to be a spanning tree of D. Let us add three additional edges to S connecting each of the vertices u_1, u_2, and u_3 with a vertex of S and let T be the resulting tree. If we define p_1 to be the unique simple path from u_1 to u_2 in T, p_2 to be the unique simple path from u_1 to u_3 in T, and, finally, u to be the last vertex on p_1 and p_2 visited by both paths, u must be a vertex of S that is 3-vertex-connected to x: No 2-vertex-cut can remove vertices from all three simple paths in T from u to u_1, from u to u_2 and from u to u_3, and, for $1 \leq i \leq 3$, no 2-vertex-cut can remove vertices from more than two disjoint paths from u_i to x in G. But as a vertex of S we know that u cannot be 3-vertex connected to x. Hence, our assumption, that three unmarked vertices are connected to a vertex of D must be wrong. □

Starting in a marked vertex v, we can determine the two unmarked vertices u_1 and u_2 that are connected to the connected component D of $G_x - \{u_1, u_2\}$ containing v by a depth-first search visiting only edges adjacent to a vertex of D. Given u_1 and u_2 we replace G and G_x according to Lemma 11 taking $C = \{u_1, u_2\}$. It is easy to see that a vertex of the remaining graph G_x is 3-vertex-connected to x if and only if this was true before the update. Hence, recursively repeating these steps for each connected component of $G_x - U$ results in a graph G with no 2-vertex-cut C of G having a connected component D of $G - C$ with $D \cap \{s_1, s_2, t_1, t_2\} = \emptyset$. Moreover, it is easy to see that the running for removing all marked vertices can be bounded by $O(m)$.

Up to now, we followed the sketch of Itai's 2P-reduction of an instance of the 2-VDPP to another instance of the 2-VDPP defined on a triconnected graph. Like Itai we started with removing cut-points in Sect. 4.2, and later removing special kinds of 2-vertex-cuts. From now on, we will no longer follow the sketch of Itai's reduction anymore, which consists of some more reduction steps removing the remaining 2-vertex-cuts. Instead, we use a special property of our own implementation of Itai's reduction algorithm: Remember that

at the beginning of Sect. 4.3 on page 14 we have inserted extra edges $(s_1, s_2), (s_1, t_2), (t_1, s_2)$, and (t_1, t_2) to our graph and note that none of the following reductions has removed these edges. These extra edges now guarantee that our graph after all reductions is triconnected:

Lemma 14. *The graph G of the instance $I = (G, s_1, s_2, t_1, t_2)$ resulting from the reductions described in this section is triconnected.*

Proof. Let us suppose that G is not triconnected. Then there exists at least one 2-vertex-cut C' of G. Let D_1 be the connected component that contains s_1. D_1 must exist, since $C' \cap \{s_1, s_2, t_1, t_2\} = \emptyset$. Since s_1 is connected to s_2 by (s_1, s_2) and to t_2 by (s_1, t_2), s_2 and t_2 are also contained in D_1 and with the same argument we can conclude that D_1 also contains t_1. Hence, there exists at least one component D_2 containing no vertex of $\{s_1, s_2, t_1, t_2\}$. But we have already removed all 2-vertex-cuts C of G such that there is a connected component D in $G - C$ with $D \cap \{s_1, s_2, t_1, t_2\} = \emptyset$. □

Thus, we have shown:

Theorem 15 (Itai, cf. Shiloach [28]). *There is a 2PR-algorithm that, given an instance $I = (G, s_1, s_2, t_1, t_2)$ of the 2-VDPP, in $O(m)$ time, either solves I or reduces I to an instance $I' = (G', s'_1, s'_2, t'_1, t'_2)$ of the 2-VDPP such that G' is a triconnected graph.*

We now might hope that on instances $I = (G, s_1, s_2, t_1, t_2)$ of the 2-VDPP, where G is a triconnected graph, there are always two disjoint paths p_1, from s_1 to t_1, and p_2, from s_2 to t_2. Unfortunately, Fig. 5 shows that this is not true.

Hence, we might therefore think of searching for a 2PR-algorithm that reduces a given instance $I = (G, s_1, s_2, t_1, t_2)$, where G is a triconnected graph, to an instance $I' = (G', s'_1, s'_2, t'_1, t'_2)$ with G' being a 4-connected graph. But before doing this, let us try to solve the 2-VDPP on 4-connected graphs:

5 Solving the 2-VDPP on 4-connected graphs

In this section we want to solve the 2-VDPP on 4-connected graphs. On 3-connected and, hence, also on 4-connected planar graphs there is a linear time algorithm for solving the 2-VDPP:

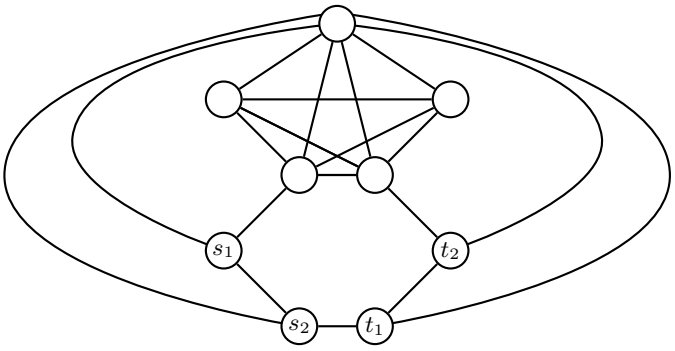

Fig. 5. A triconnected graph without a solution for the 2-VDPP.

Theorem 16 (Perl and Shiloach, [20]). *On a triconnected planar graph the 2-VDPP is solvable in linear time.*

The interested reader will find a very simple and short proof of Theorem 16 in a paper of Woeginger [37].

We now to sketch how and why Shiloach's algorithm [28] always finds a solution of the 2-VDPP on 4-connected nonplanar graphs. However, the reader only needs to know the Theorems 16 and 20 to understand the following sections of this paper. Thus, the reader may notice these theorems and skip the rest of this section.

Using Definition 17 and 18 we will give in Lemma 19 a well-known characterization of planar graphs, which, implicitly, also characterizes nonplanar graphs.

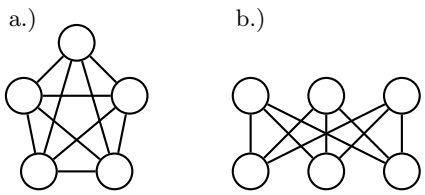

Fig. 6. a.) The K_5 and b.) the $K_{3,3}$.

Definition 17. *The K_5 is the undirected graph without any loop*[5] *that consists of five vertices and edges between every pair of different vertices (cf. Fig. 6.a). The $K_{3,3}$ is the undirected graph that consists of two sets S_1 and S_2 of three vertices and edges between every vertex of S_1 and every vertex of S_2, but has no other edges (see Fig. 6.b).*

Definition 18. *A graph G_1 is a subdivision of a graph G_2 if G_1 is obtained from G_2 by replacing each edge (u, v) by a path from u to v such that the paths for each pair of distinct edges apart from at most one common endpoint are disjoint.*

Lemma 19 (Kuratowski [14]). *An undirected graph is planar if and only if it contains no subgraph that is a subdivision of K_5 or $K_{3,3}$.*

It is very easy to solve the 2-VDPP on a 4-connected nonplanar graph G containing K_5 as a subgraph: First, we construct four disjoint paths from s_1, s_2, t_1, t_2 to four pairwise distinct endpoints of the K_5. Such paths must exist due to Corollary 6. We then just connect the endpoints $k_1 \in K_5$ and $k_2 \in K_5$ of the constructed paths starting in s_1 and t_1, respectively, by edge (k_1, k_2), and, similarly, connect the endpoints $k_3 \in K_5$ and $k_4 \in K_5$ of the paths starting in s_2 and t_2, respectively, by edge (k_3, k_4). The resulting paths solve our instance of the 2-VDPP. We can use nearly the same idea to find two disjoint paths solving an instance of the 2-VDPP if G contains $K_{3,3}$ as a subgraph (instead of connecting the endpoints by just one edge we may need two edges to connect the endpoints. Moreover, in this case, w.l.o.g. we should assume that before reaching an endpoint in $K_{3,3}$ each of the constructed disjoint paths from the vertices in $\{s_1, s_2, t_1, t_2\}$ to four vertices $k_1, k_2, k_3, k_4 \in K_{3,3}$ should not visit any other vertex of the $K_{3,3}$).

If a 4-connected nonplanar graph G does not contain K_5 or $K_{3,3}$ as subgraphs, but contains a subgraph which is a subdivision of K_5 or $K_{3,3}$, constructing disjoint paths is somewhat more complicated: For example, have a look at the graph of Fig. 7.a. It is easy to see that there are paths consisting of the dotted edges in Fig. 7.a between each pair of v_1, w_1, v_2, w_2, and z such that the paths for

[5] A loop is an edge having the same vertex as its endpoints.

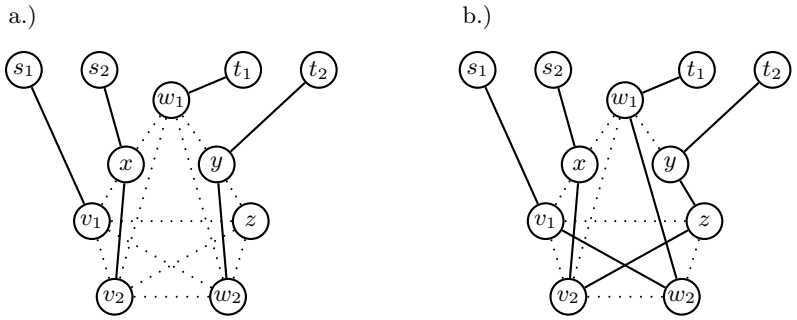

Fig. 7. a.) A routing problem and b.) its solution if G contains a subdivision of K_5.

different pairs of vertices have at most one of their endpoints in common. We can conclude that the subgraph induced by the vertices v_1, w_1, v_2, w_2, x, y, and z is a subdivision of K_5. As above, let us start with constructing four disjoint paths from the vertices s_1, s_2, t_1, and t_2 to the vertices v_1, v_2, w_1, and w_2 as shown in Fig. 7.a (the solid lines represent the constructed paths). We now have the problem that v_1 and w_1 are not directly connected by an edge, but only by a path like the path consisting of the edges (v_1, x) and (x, w_1) with x being visited by the one of the four constructed paths leading from s_2 to v_2. A more precise look at Fig. 7.a shows that the four constructed paths from s_i to v_i and t_i to w_i $(1 \leq i \leq 2)$, can not be extended to two disjoint paths leading from s_1 to t_1 and from s_2 to t_2. Nevertheless, there may be another routing defining two disjoint paths as shown in Fig. 7.b. Indeed, Watkins [34] showed that in a 4-connected nonplanar graph containing a subdivision of K_5 each instance of the 2-VDPP has a solution. Shiloach [28] proved that the same is true of 4-connected nonplanar graphs containing a subgraph that is a subdivision of $K_{3,3}$. Moreover, given an instance of the 2-VDPP on a nonplanar 4-connected graph containing a subdivision of K_5 or $K_{3,3}$ and given this subdivision, the algorithms of Watkins and Shiloach determine in linear time two paths solving the instance

of the 2-VDPP. Hopcroft and Tarjan showed [8] that it is possible to test whether a given graph contains a subgraph that is a subdivision of $K_{3,3}$ or K_5 in linear time and Wiliamson [36] showed that in this case such a subgraph can be determined again in linear time. A more precise look at Shiloach's and Watkins' algorithm shows that G does not really need to be 4-connected. As already observed by Shiloach in [28] the algorithms of Shiloach and Watkins work on all triconnected nonplanar graphs in which there are four disjoint paths between s_1, s_2, t_1, t_2 and every set S of at most four vertices (in the case of $|S| \leq 3$ we allow the endpoints of the four paths in S to overlap). We can conclude:

Theorem 20 (Shiloach [28]). *Let $G = (V, E)$ be a triconnected nonplanar graph for which there are four disjoint paths between s_1, s_2, t_1, t_2 and every set $S \subseteq V - \{s_1, s_2, t_1, t_2\}$ with $|S| \leq 4$, except that in the case of $|S| < 4$ the endpoints of the paths in S may overlap. Then, the 2-VDPP on (G, s_1, s_2, t_1, t_2) can be solved in linear time.*

For a detailed proof have a look at the papers of Shiloach [28], of Watkins [34], and of Wiliamson [36].

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

As we have already seen in Sect. 5, 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 2P-reduce I, in $O(m\alpha(m, n))$ time, to 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). For this purpose we will make use of the following lemma, which was implicitly proven in reduction $R3$ of Shiloach's paper [28].

Lemma 21 (Shiloach, [28]). *Let $I = (G, s_1, s_2, t_1, t_2)$ be an instance of the 2-VDPP, where $G = (V, E)$ is a triconnected graph. If there is a set $S \subseteq V - \{s_1, s_2, t_1, t_2\}$ with at most four vertices*

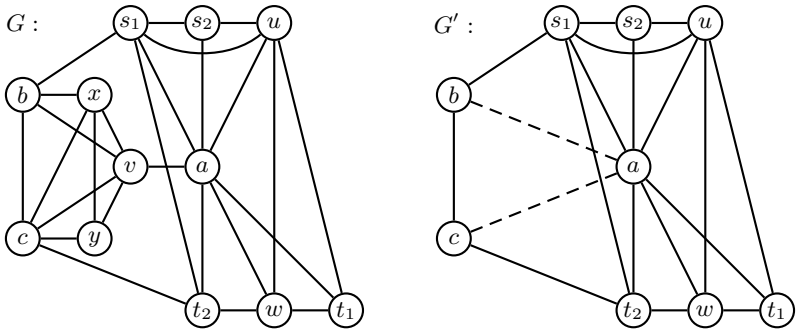

Fig. 8. A graph G and the Δ-replacement G' of G by the 3-vertex-cut $C = \{a, b, c\}$ removing v. The dashed lines represent newly inserted edges.

such that there are no four disjoint paths from s_1, s_2, t_1, and t_2 to S, where the endpoints of the paths are allowed to overlap, then there is a 3-vertex-cut C of G that separates at least one vertex $v \in V$ from $\{s_1, s_2, t_1, t_2\}$.

From Lemma 21 it follows that, if G contains no triangular cut separating a vertex v from $\{s_1, s_2, t_1, t_2\}$, there are four paths from s_1, s_2, t_1, and t_2 to any other subset $S \subseteq V - \{s_1, s_2, t_1, t_2\}$ and, hence, our instance of the 2-VDPP can be solved in linear time on G. For reducing our original triconnected graph G to a graph without any triangular cut separating a vertex v from $\{s_1, s_2, t_1, t_2\}$, we 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 exist vertices $v_1, v_2, \ldots, v_l$ in G and a 3-vertex-cut $C = \{a, b, c\}$ that separates each of the vertices $v_1, \ldots, v_l$ from $\{s_1, s_2, t_1, t_2\}$. Like Gustedt in [7], we call a graph G' a *triangle-replacement of G by C (removing the vertices $v_1, \ldots, v_l$)* or, for short, a *Δ-replacement of G by C* if it is obtained from G by deleting all vertices of the connected components of $G - C$ containing the vertices $v_1, \ldots, v_l$ together with their adjacent edges and by inserting edges between all pairs of vertices of C that are not already connected by an edge (see Fig. 8). Sometimes we also call the replacement step that replaces G by G' a Δ-replacement.

The reader may have noticed that a Δ-replacement seems to be very similar to the replacement given Lemma 11. Indeed, we try to

delete all 3-vertex-cuts separating a vertex v from $\{s_1, s_2, t_1, t_2\}$ in the same way as we have deleted 2-vertex-cuts separating a vertex v from $\{s_1, s_2, t_1, t_2\}$ in Section 4.3, except that the connectivity degree is increased by 1. Some lemmas of Section 4.3 will have corresponding lemmas in this section. However, one fact implicitely used in Section 4.3 and making it easy to remove 2-vertex-cuts from G does not hold for 3-vertex-cuts: If after the replacement of G according to Lemma 11 we search for $k \leq 3$ internally disjoint paths between to vertices u and v we know that such paths exist if and only if such paths exist before the update, whereas a corresponding fact for $k \leq 4$ internally disjoint paths between to vertices a Δ-replacement does not hold. Thus, we have to find out in much more detail in which cases the existence or noexistence of disjoint paths are preserved after a Δ-replacement.

Let us define a set of k paths $p_1, \ldots, p_k$ to be *quasi internally disjoint* (or, for short, *q.i. disjoint*) if they are pairwise edge-disjoint and, for all $i, j \in \{1, \ldots, k\}$ with $i \neq j$, no inner vertex of p_i appears on p_j. Then, Lemma 22 describes in which cases the existence of disjoint paths p_i from a vertex u_i to a vertex v_i is preserved after a series of Δ-replacements by 3-vertex-cuts $C_1, C_2, \ldots, C_l$. The condition, that, for all $u_i \in \bigcup_{r=1}^{l} C_r$, there is no $j \neq i$ with $u_i = u_j$ is chosen in such a way, that it includes the case of two disjoint paths from s_1 to t_1 and s_2 to t_2 with two or more vertices lying in one of the 3-vertex-cuts $C_1, C_2, \ldots, C_l$, but, in order to guarantee the correctness of the lemma, excludes the case where $u_1 = \ldots = u_k$ and $v_1 = \ldots = v_k$ and v_1 and u_1 are vertices of the same 3-vertex-cut C_i .

Lemma 22. *Let $G_1 = (V_1, E_1)$ be a triconnected graph and $G_2 = (V_2, E_2)$ be a graph obtained by a series of Δ-replacements of G_1 by 3-vertex-cuts $C_1, C_2, \ldots, C_l$ in this order. Let $u_1, \ldots, u_k$, $v_1, \ldots, v_k$ be vertices of V_2 with $\{u_1, \ldots, u_k\} \cap \{v_1, \ldots, v_k\} = \emptyset$. If, for all $u_i \in \bigcup_{r=1}^{l} C_r$, there is no $j \neq i$ with $u_i = u_j$, and if there are k q.i. disjoint paths p_i $(1 \leq i \leq k)$ in G_1 leading from u_i to v_i, there are also k q.i. disjoint paths q_i from u_i to v_i in G_2, where q_i visits a subset of the vertices visited by p_i. Conversely, if $u_1, \ldots, u_k, v_1, \ldots, v_k$ are pairwise distinct and if there are k disjoint paths q_i $(1 \leq i \leq k)$ in G_2 leading from u_i to v_i, then there are also k disjoint paths p_i $(1 \leq i \leq k)$*

in G_1 leading from u_i to v_i. Given, for each Δ-replacement with 3-vertex-cut C_j, the vertices of C_j as well as a vertex x_j removed by this Δ-replacement, such paths $p_1, \ldots, p_k$ can be constructed from $q_1, \ldots, q_k$ in $O(|V_1| + |E_1|)$ time.

Proof. First, assume that, for all $1 \leq i \leq k$ with $u_i \in \bigcup_{r=1}^{l} C_r$, there is no j with $1 \leq j \leq k$ and $i \neq j$ such that $u_i = u_j$. Let $p_1, \ldots, p_k$ be k q.i. disjoint paths in G_1 leading from u_i to v_i $(1 \leq i \leq k)$. We first prove that, for $0 \leq j \leq l$, the following is true: The graph H_j obtained from the Δ-replacements of G_1 by the 3-vertex-cuts $C_1, \ldots, C_j$ in this order contains k q.i. disjoint paths $p'_1, \ldots, p'_k$ with p'_i $(1 \leq i \leq k)$ leading from u_i to v_i and visiting a subset of the vertices visited by p_i.

Clearly, the assertion holds for $j = 0$. Let us now assume that the assertion holds for an arbitrary $j \in \{0, \ldots, l-1\}$, i.e. there are k q.i. disjoint paths $p'_1, \ldots, p'_k$ in H_j with the properties described above. If none of the paths $p'_1, \ldots, p'_k$ visits an edge outside of graph H_{j+1} the assertion holds also for $j+1$. Otherwise, at least one path p of the paths $p'_1, \ldots, p'_k$ uses an edge (a, b) that is deleted from H_j after the Δ-replacement of H_j by C_{j+1}. p must also visit a vertex $c \in C_{j+1}$ before following the edge (a, b) and a vertex $d \in C_{j+1}$ after having reached (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 (here we use the fact that for each $u_i \in \bigcup_{r=1}^{l} C_r$ there is no $i \neq j$ with $u_i = u_j$). We may need to repeat this step one more time in order to obtain k pairwise q.i. disjoint paths $p''_1, \ldots, p''_k$ such that no path uses an edge that is deleted from H_j after the Δ-replacement of H_j by C_{j+1}, and such that the vertices visited by p''_i are a subset of the vertices visited by p'_i and, hence, a subset of the vertices visited by p_i. After this replacement it is easy to see that the invariant also holds for $j+1$.

Conversely, let us assume that $u_1, \ldots, u_k, v_1, \ldots, v_k$ are pairwise distinct and that there are k disjoint paths $q_1, \ldots, q_k$ in G_2 with q_i $(1 \leq i \leq k)$ leading from u_i to v_i. If one of the paths uses an edge $e = (a, b)$ with $e \notin E_1$ that was inserted by the Δ-replacement with 3-vertex-cut C_j, we replace this edge by a *replacement* path constructed as follows:

Let $H_{j-1} = (V', E')$ and $H_j = (V^*, E^*)$ again be the graphs obtained from the Δ-replacements of G_1 by the 3-vertex-cuts $C_1, \ldots,$ C_{j-1} or $C_1, \ldots, C_j$, respectively. Moreover, let x_j be a vertex removed by the Δ-replacement with 3-vertex-cut C_j. Construct with standard network flow techniques three disjoint paths q'_1, q'_2, and q'_3 in H_{j-1} starting from x_j, but each leading to its own endpoint in C_j (such paths must exist, since they exist in the triconnected graph $H_0 = G_1$ and since we have already shown that therefore the same must be true of graph H_{j-1}). q'_1, q'_2, and q'_3 visit only edges in $E' - E^*$. Otherwise, before visiting an edge in E^*, they would visit a vertex in C_j, and hence they cannot be both simple and disjoint (remember that we have defined paths to be simple paths). We define the replacement path for edge (a, b) to be the path from a to b resulting from the concatenation of the two paths of q'_1, q'_2, and q'_3 leading from the common vertex x_j to either a or b, respectively (the path from x_j to a should be visited in reverse direction).

Note that for fixed j, due to the simplicity of $q_1, \ldots, q_k$ no paths of $q_1, \ldots q_k$ can visit more than two of the edges inserted by the Δ-replacement with 3-vertex-cut C_j, and if there is a path visiting two such edges e_1 and e_2, e_1 must be visited immediately before or immediately after e_2. Hence, in this case the two edges can be replaced by the third edge e_3 between the vertices of C_j, and in a next step by the replacement path for e_3. This guarantees that after replacing edges by their replacement paths the paths are still simple.

We have to show that even after recursively replacing all edges $e \notin E_1$ by their replacement paths the resulting k paths $p_1, \ldots, p_k$ are disjoint. Note that the replacement paths for the new edges inserted by one Δ-replacement are not disjoint. This is not necessary, since only one of k disjoint paths from u_i to v_i $(1 \leq i \leq k)$ can visit one or two of the three newly inserted edges. However, replacement paths constructed in different Δ-replacements are disjoint since the edges of the replacement paths are deleted after the Δ-replacement for which they have been constructed. Hence, the paths obtained by replacing each edge $e \notin E_1$ by its replacement path must be disjoint.

Finally, note that the construction of a replacement path is dominated by the time needed to determine the three disjoint paths from x_j to the different vertices of C_j on the subgraph of H_{j-1} that is deleted after the Δ-replacement of H_{j-1} by C_j. Hence, we obtain a

running time of $O(|V_1| + |E_1|)$ for replacing all edges $e \notin E_1$ by their replacement paths. □

In order to delete all 3-vertex-cuts separating a vertex v from $\{s_1, s_2, t_1, t_2\}$, we repeatedly search for such a vertex v of G separated from $\{s_1, s_2, t_1, t_2\}$ by a 3-vertex-cut C and replace I by $I^* = (G^*, s_1, s_2, t_1, t_2)$, where G^* is the Δ-replacement of G by C 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 3-vertex-cut C separating a vertex v from $\{s_1, s_2, t_1, t_2\}$.

This idea will only work, if the Δ-replacements do not change the solvability of the problem.

Lemma 23. *[Shiloach, [28]] 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.*

This lemma is similar to Lemma 11 in Section 4.3. In Shiloach's paper [28] it is proven directly following the ideas used in Lemma 11, but it also follows as a corollary from Lemma 22.

Proof. Let $C = \{a, b, c\}$ be chosen in such a way that G^* is the Δ-replacement of G by C. For two vertices of C, say w.l.o.g. a and b, it is easy to see that there are three internally disjoint paths from a to b in G^*. The first path consists of edge (a, b), the second of the edges (a, c) and (c, b) and the third of the replacement path of edge (a, b) as defined in the proof of Lemma 22. Finally, from Lemma 22 with $G_1 = G$ and $G_2 = G^*$ it follows that there are three disjoint paths between all other pairs of vertices of V^* and that there are two disjoint paths from s_1 to t_1 and s_2 to t_2 in G iff the same is true in G^*. □

For efficiently supporting the construction of disjoint paths from s_1 to t_1 and from s_2 to t_2 in our original graph G from two disjoint paths in G', i.e. the graph G after a series of Δ-replacements removing all triangular cuts separating a vertex v from $\{s_1, s_2, t_1, t_2\}$, we store with each newly inserted edge (a, b) that is inserted by a Δ-replacement of G with 3-vertex-cut C the vertices of C as well as a vertex v removed by the Δ-replacement.

Similar as in Section 4.3 we can conclude:

Lemma 24 (Shiloach, [28]). *Let $\mathcal{A}$ be the algorithm consisting of the repeated construction of Δ-replacements and storing with each newly inserted edge e a 3-vertex-cut C and a vertex v such that e is inserted by the Δ-replacement with 3-vertex-cut C removing v. Then, $\mathcal{A}$ is a 2PR-algorithm.*

Since in Shiloach's paper [28] an explicit proof of this fact is given only for the case of one Δ-replacement, we want to prove Lemma 24 in this paper.

Proof. Let $G = (V, E)$ be the graph on which $\mathcal{A}$ is started and $G' = (V', E')$ be the graph that is output by $\mathcal{A}$. Since a Δ-replacement deletes at least one vertex of G and inserts at most three new edges, it is easy to see that $|V'| \leq |V|$ and $|E'| = O(|E|)$. The rest follows from Lemma 22. □

From Lemma 24 in combination with Lemma 21 and Theorem 20 it follows that after reducing I to an instance $I' = (G', s'_1, s'_2, t'_1, t'_2)$ such that in G' there is no triangular cut separating a vertex v from $\{s'_1, s'_2, t'_1, t'_2\}$, the 2-VDPP on I' can be solved in linear time. We only need to show that the reduction from I to I' can be implemented efficiently. Therefore, we only need to find efficient algorithms for the following problems:

- Find a vertex v that is separated from $\{s_1, s_2, t_1, t_2\}$ by a 3-vertex-cut.
- Find a 3-vertex-cut C separating v from $\{s_1, s_2, t_1, t_2\}$.
- Given v and C, replace G by the Δ-replacement of G by C.

Let us start with the latter problem:

Lemma 25. *Given an instance $I = (G, s_1, s_2, t_1, t_2)$ of the 2-VDPP, where $G = (V, E)$ is a triconnected graph, a triangular cut C of G, and a vertex $v \in V$ which is separated from $\{s_1, s_2, t_1, t_2\}$ by C, the Δ-replacement $G^* = (V^*, E^*)$ of G by C removing v can be constructed in $O(|E - E^*|)$ time.*

Proof. It is easy to see that G^* can be constructed by a depth-first search on G starting in v and not visiting any edge of E^*. □

According to Lemma 25 it follows that the running time for the construction of all Δ-replacements can be bounded by $O(m)$. Thus, our problem of efficiently solving the 2-VDPP reduces to the problem of efficiently determining a vertex v and a 3-vertex-cut C such that v is separated from $\{s_1, s_2, t_1, t_2\}$ by C. The main difference between the algorithm of this paper and Shiloach's algorithm is the computation of such vertices and 3-vertex-cuts. While not searching explicitly for 3-vertex-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 restarted on the resulting graph. Since this may happen $\Theta(n)$ times, the running time of Shiloach's algorithm is bounded only by $O(mn)$.

Here, in contrast, we systematically remove all 3-vertex-cuts separating a vertex v from $\{s_1, s_2, t_1, t_2\}$ efficiently. For identifying such a vertex v, like in Section 4, we 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 12 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 3-vertex-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). Unfortunatley, whereas in Section 4.3, after a replacement as it was decribed in Lemma 11, two vertices u and w are 3-vertex-connected if and only the same was true before the replacement, the existence or nonexistence of 4 internally disjoint paths between two vertices may not be preserved by a Δ-replacement. 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 26. *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 and let G^* be a Δ-replacement of G. Let G_x and G_x^* be defined as above. Then, if x and $b \in V^*$ are 4-connected in G_x, they are also 4-connected in G_x^*.*

Lemma 26 follows directly from Lemma 22 (note that G_x^* can be considered as a Δ-replacement of G_x). 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 think of using a *dynamic data structure*. This is a data structure that supports two kinds of operations: update operations, which in the 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 [10] 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 (see Fig. 9). 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 27. Queries in H_x can now be answered with the data structure of Kanevsky et al.

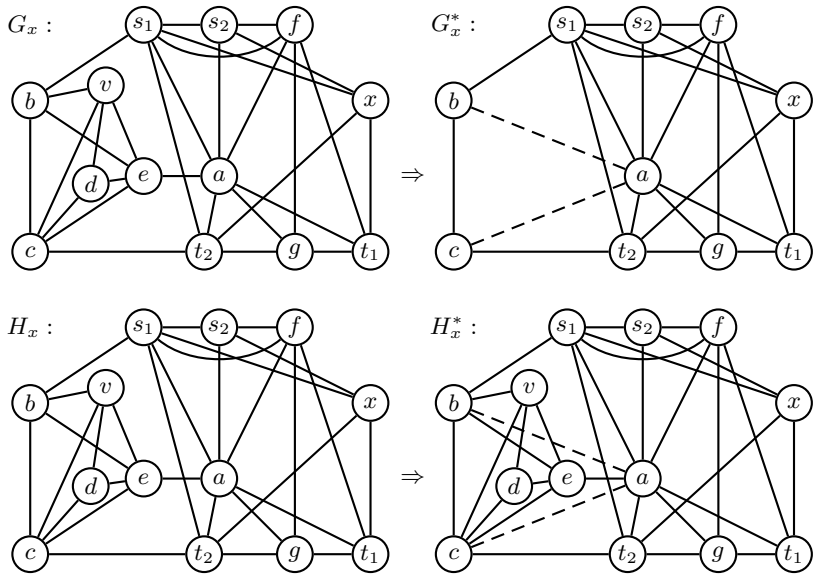

Fig. 9. The changes in G_x and H_x due to the Δ-replacement of G_x by $\{a, b, c\}$. The dashed lines represent newly inserted edges.

Lemma 27. *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. If a vertex of G_x is 4-connected to x in G_x it must be 4-connected to x in H_x since G_x is a subgraph of H_x. The reverse direction follows directly from Lemma 22 (note that we can consider G_x as a graph obtained from H_x by a series of Δ-replacements). □

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-vertex-cut C that separates v and x. Remember that finding a 2-vertex-cut separating a vertex v from $\{s_1, s_2, t_1, t_2\}$ in Section 4.3 was very easy. We could show that there exist exactly 2-vertices that are connected by an edge to the connected component containing v in the graph obtained from G_x be deleting all vertices that are 3-vertex-connected to x. Unfortunately, a corresponding lemma does not hold for Δ-replacements. However, once again, instead of searching for a 3-vertex-cut of G_x, we search for a 3-vertex-cut of H_x:

Lemma 28. *Let v be a vertex of G_x and let C be a 3-vertex-cut separating v and x in H_x. Then C is also a 3-vertex-cut 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-vertex-cut $C \subseteq V_{G_x^*}$ in H_x^* that separates x and a vertex $w \in V_{G_x^*}$ also separates x and w 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-vertex-cut C with $C \not\subseteq V_{G_x^*}$ that separates x and a vertex $w \in V_{G_x^*}$ 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 it follows from Lemma 22, 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 C is not visited by q_1, q_2, and q_3. But this contradicts Corollary 7. □

For determining a 3-vertex-cut 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 C_1 and C_2 such that one of these sets is a 3-vertex-cut separating u and w (see [10] for more details). Now, for two sets C_1 and C_2 with one of them being a 3-vertex-cut separating v and x, we start two interleaved depth-first searches on $G_x - C_1$ and on $G_x - C_2$ with v as the source vertex; 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 $G_x - C_1$ or to that on $G_x - C_2$, either C_1 (in the first case) or C_2 (in the second case) is a 3-vertex-cut 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-vertex-cut $C = C_1$ or $C = C_2$, and, after identifying C, 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 C, the extra running time for

determining 3-vertex-cuts 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-vertex-cuts 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 and removing 3-vertex-cuts run in linear time, we can conclude that the following lemma holds:

Theorem 29. *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 and triconnected graph. Then, in $O(m\alpha(m,n))$ time, I can be 2P-reduced to an instance $I' = (G', s_1, s_2, t_1, t_2)$ of the 2-VDPP 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 3-vertex-cut.*

In combination with Theorem 20 and Lemma 21 we can conclude:

Theorem 30. *The 2-VDPP can be solved in $O(n+m\alpha(m,n))$ time.*

7 Extensions

Perl and Shiloach [20] 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 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 p_1 from s_1 to t_1 and p_2 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 [20] 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.

As shown by the author of this paper [33], a well known reduction of Lucchesi and Giglio [15] for the decision version of the 2-VDPP on dags can be generalized to reduce the (general) version of the 2-VDPP on dags in $O(m \log_{2+m/n} n + n \log^3 n)$ time to the 2-VDPP on undireted graphs. With the results of this paper we obtain an $O(m \log_{2+m/n} n + n \log^3 n)$-time algorithm for solving 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 [29]. Hence, the 2-EDPP on dags is also solvable in $O(m \log_{2+m/n} n + n \log^3 n)$ time.

8 Acknowledgments

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 pre-versions of this paper. Finally the author thanks the anonymous refeeres for many suggestions improving the readability of this paper. Especially the approach for deleting vertex-cuts separating a vertex from $\{s_1, s_2, t_1, t_1\}$ helped to make the correponding section shorter and easier to understand.

References

1. J. Bang-Jensen and G. Gutin, *Digraphs: Theory, Algorithms and Applications*, Springer, London, 2001.
2. G. Di Battista and R. Tamassia, On-line maintenance of triconnected components with SPQR-trees. *Algorithmica* **15** (1996), pp. 302–318.
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. J. E. Hopcroft and R. E. Tarjan, Efficient planarity testing, *J. ACM* **21** (1974), pp. 549–568.
9. D. Jungnickel, *Graphs, Networks and Algorithms (2nd ed.)*, Springer, Berlin, 2005.
10. 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.
11. R. M. Karp, On the computational complexity of combinatorial problems, *Networks* **5** (1975), pp. 45–68.
12. 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.
13. E. Korach and A. Tal, General vertex disjoint paths in series-parallel graphs, *Discrete Appl. Math.* **41** (1993), pp. 147–164.
14. K. Kuratowski, Sur le probléme des courbes gauches en topologie, *Fund. Math.* **15**, pp. 271-283.
15. 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.
16. J. F. Lynch, The equivalence of theorem proving and the interconnection problem, *(ACM) SIGDA Newsletter,* **5** (1975), pp. 31–36.
17. K. Menger, Zur allgemeinen Kurventheorie. *Fund. Math.* **10** (1927), pp. 96–115.
18. 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.
19. 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.
20. Y. Perl and Y. Shiloach, Finding two disjoint paths between two pairs of vertices in a graph, *J. ACM* **25** (1978), pp. 1–9.
21. N. Robertson and P. D. Seymour, Graph minors. XIII. The disjoint paths problem, *J. Comb. Theory, Ser. B*, **63** (1995), pp. 65–110.
22. 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.

23. A. Schrijver, A group-theoretical approach to disjoint paths in directed graphs, *CWI Quarterly* **6** (1993), pp. 257–266.
24. A. Schrijver, *Combinatorial optimization – Polyhedra and Efficiency Vol. A*, Springer, Berlin, 2002.
25. A. Schrijver, *Combinatorial optimization – Polyhedra and Efficiency Vol. C*, Springer, Berlin, 2002.
26. 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.
27. P. D. Seymour, Disjoint paths in graphs, *Discrete Math.* **29** (1980), pp. 293–309.
28. Y. Shiloach, A polynomial solution to the undirected two paths problem, *J. ACM* **27** (1980), pp. 445–456.
29. J. W. Suurballe and R. E. Tarjan, A quick method for finding shortest pairs of disjoint paths. *Networks* **14** (1984), pp. 325-336.
30. R. E. Tarjan and J. van Leeuwen, Worst-case analysis of set union algorithms, *J. ACM* **31** (1984), pp. 245–281.
31. C. Thomassen, 2-linked graphs, *Europ. J. Combinatorics* **1** (1980), pp. 371–378.
32. 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.
33. T. Tholey, Finding Disjoint Paths on Directed Acyclic Graphs, Report 2005-04, Universität Augsburg, Institut für Informatik, 2005.
34. M. E. Watkins, On the existence of certain disjoint arcs in graphs, *Duke Math. J.* **35** (1968), pp. 231–246.
35. H. Whitney, Congruent graphs and the connectivity of graphs, *Amer. J. Math.* **54** (1932), pp. 150–168.
36. S. G. Williamson, Depth-first search and Kuratowski subgraphs, *J. ACM* **31** (1984), pp. 681–693.
37. G. Woeginger, A simple solution to the two paths problem in planar graphs, *Inform. Process. Lett.* **36** (1990), pp. 191–192.