Discrete Applied Mathematics 166 (2014) 263–268

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Note

Algorithms for finding a rooted $(k, 1)$-edge-connected orientation

Csaba Király *

Department of Operations Research, Eötvös Loránd University, H-1117 Budapest, Pázmány Péter sétány 1/C, Hungary

ARTICLE INFO

Article history:
Received 21 March 2013
Received in revised form 26 September 2013
Accepted 1 October 2013
Available online 1 November 2013

Keywords:
Orientation
Rooted edge-connectivity
Highly k-tree-connected

ABSTRACT

A digraph is called rooted $(k, 1)$-edge-connected if it has a root node r_0 such that there exist k arc-disjoint paths from r_0 to every other node and there is a path from every node to r_0. Here we give a simple algorithm for finding a $(k, 1)$-edge-connected orientation of a graph. A slightly more complicated variation of this algorithm has running time $O(n^4 + n^2m)$ that is better than the time bound of the previously known algorithms. With the help of this algorithm one can check whether an undirected graph is highly k-tree-connected, that is, for each edge e of the graph G, there are k edge-disjoint spanning trees of G not containing e. High tree-connectivity plays an important role in the investigation of redundantly rigid body–bar graphs.

1. Introduction

All graphs and digraphs considered here are loopless but may contain parallel edges, and k and ℓ are always positive integers with $k \geq \ell$. The number of vertices and edges in a graph will be denoted by n and m, respectively. A digraph $D = (V, A)$ with a root node $r_0 \in V$ is called *r_0-rooted (k, ℓ)-edge-connected* if $\delta_D(X) \geq k$ and $\varrho_D(X) \geq \ell$ for every set $X \subset V$ with $r_0 \in X$, where $\delta_D(X)$ denotes the out-degree of X and $\varrho_D(X)$ denotes the in-degree of X. By Menger's theorem this is equivalent to the following: there are k arc-disjoint paths from r_0 to every other node and there are ℓ arc-disjoint paths from every node to r_0. A rooted $(k, 0)$-edge-connected digraph is called *rooted k-edge-connected*.

An undirected graph $G = (V, E)$ is called *(k, ℓ)-partition-connected* if

$$e_G(\mathcal{P}) \geq k(|\mathcal{P}| - 1) + \ell$$

for every partition $\mathcal{P}$ of V, where $e_G(\mathcal{P})$ denotes the number of edges that are not induced by any set of the partition. A $(k, 0)$-partition-connected graph is called *k-partition-connected*. Tutte's theorem [13] implies that a graph $G = (V, E)$ is (k, ℓ)-partition-connected for positive integers k and ℓ if and only if, for every at most ℓ edges of G, there are k edge-disjoint spanning trees of G avoiding these ℓ edges. When $\ell = 1$ this property is called *high k-tree-connectivity*. Connelly, Jordán and Whiteley [1] proved that the "body–bar graph" G_H of a graph H is "redundantly rigid" in $\mathbb{R}^d$ if and only if H is highly $\binom{d+1}{2}$-tree-connected. This result motivates us to give a simple algorithm for testing high k-tree-connectivity.

From the result of Frank [5] it is easy to show that (k, ℓ)-partition-connectivity and (k, ℓ)-edge-connected orientability are equivalent. (To prove this, just use the main theorem of the paper with the following function: $l(X) = k$ if $r_0 \notin X \neq \emptyset$, $l(X) = \ell$ if $r_0 \in X \neq V$, $l(X) = 0$ if $X \in \{\emptyset, V\}$.)

Theorem 1.1. *For k, ℓ integers with $k \geq \ell$ a graph with a root node r_0 has an r_0-rooted (k, ℓ)-edge-connected orientation if and only if it is (k, ℓ)-partition-connected.* □

* Tel.: +36 702853293; fax: +36 13812174.
E-mail address: cskiraly@cs.elte.hu.

0166-218X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.dam.2013.10.004

The proof of [5] gives rise to an algorithm for finding a rooted (k, ℓ)-edge-connected orientation (and by this for testing (k, ℓ)-partition-connectivity); the algorithm is rather involved. For the case where $\ell = 0$, when we want to give a rooted k-edge-connected orientation of a graph, the proof presented in [4] and in [6, Section 9.1] gives rise to a simpler algorithm. Yet another algorithm is given by Gabow and Manu in [9]. These algorithms output either a rooted k-edge-connected orientation of the input graph $G = (V, E)$ or a partition $\mathcal{P}$ of V for which

$$e_G(\mathcal{P}) < k(|\mathcal{P}| - 1),$$

showing that G is not k-partition-connected.

A more efficient algorithm for finding a (k, ℓ)-edge-connected orientation can be read out from the proof of Theorem 1.1 given implicitly in [6,7] that first gives the proper orientation through its in-degree vector and then uses the orientation lemma of Hakimi [10] (that has an algorithmic proof) to realize it. Though not explicitly stated in [7], the complexity of this algorithm is $O(n^5 + n^2m)$. The involved part of this algorithm is that it uses the bi-truncation algorithm of Frank and Tardos [8,12].

Moreover, this idea of using the orientation lemma leads to a more efficient algorithm for the case where $\ell = 0$ because then only the truncation algorithm of Frank and Tardos [8] is needed (see [11] and [6, Section 15.4.4]); thus we get a bound $O(n^4 + n^2m)$ for the running time. Since we use the truncation algorithm instead of the bi-truncation algorithm, this algorithm is less involved. The previously mentioned algorithm of Gabow and Manu [9] has a time bound of $O(\min\{n, \log(N)\}n^2m^* \log(n^2/m^*))$ where m^* is the edge number of the underlying simple graph and N is the maximum number of parallel copies of an edge.

Although our main aim is to give a simple and efficient algorithm for the case of $\ell = 1$, we begin by giving a new algorithm for the case of $\ell = 0$ as this algorithm will be used as a subroutine in the latter algorithms. This algorithm, that arises from modifying Frank's earlier algorithm [4] for finding rooted k-edge-connected orientations – using a simple data structure – has a time bound of $O(n^4 + n^3k)$. Next, we turn to giving a new proof of Theorem 1.1 for the case where $\ell = 1$. This proof shows a simple algorithm for testing high k-tree-connectivity. The algorithm uses the rooted k-edge-connected orientation algorithm as a subroutine. (This algorithm can be extended to hypergraphs.) Finally, by combining the ideas of these two algorithms, we obtain another algorithm for finding a $(k, 1)$-edge-connected orientation of graphs that runs in time $O(n^4 + n^3k)$.

We note that to have a (k, ℓ)-edge-connected orientation (for $k \geq \ell$), the graph needs a maximum of $2k$ parallel edges between two nodes, and thus $m = O(n^2k)$. However, at some points this can be reduced to the edge number of the underlying simple graph, that is, to a maximum of $O(n^2)$. To achieve this in the following algorithms, the graph (respectively, its orientation) will be stored via its *adjacency matrix* $((a_{i,j}))$ where $a_{i,j}$ is the number of ij-edges (respectively, arcs).

2. A more efficient algorithm for rooted k-edge-connected orientability

As noted before, there exists an algorithm for finding a rooted k-edge-connected orientation of a graph that has a running time of $O(n^4 + n^2m)$. However, this algorithm uses a polyhedral technique. Here we describe another algorithm based on Frank's algorithm [4] with a running time of $O(n^4 + n^3k)$ that uses only basic graph theoretical arguments.

First we sketch Frank's algorithm:

Algorithm 2.1 (*Frank [4]*)**.** Input: A graph $G = (V, E)$ and a root $r_0 \in V$.

Output: $\overrightarrow{G} = (V, \overrightarrow{E})$, an r_0-rooted k-edge-connected orientation of G OR a partition $\mathcal{P}$ of V with $e_G(\mathcal{P}) < k(|\mathcal{P}| - 1)$.

Phase 1: We add new edges to the graph from the root r_0 to some vertices so as to obtain a rooted k-edge-connected orientation $D = (V, A)$ of the extended graph with $\varrho_D(r_0) = 0$. More precisely, we add k parallel r_0v-edges for every $v \in V - r_0$, we orient all r_0v-edges towards v for every $v \in V - r_0$ and we orient the remaining edges arbitrarily.

Phase 2: We try to omit one by one the newly added arcs from D in such a way that the rooted k-edge-connected orientation is preserved. If the omission of a new arc r_0t decreases the in-degree of a $t\bar{r_0}$-*set* (that is, a set containing t but avoiding r_0) below k, then we reverse a path starting at t and ending at a node $v \in V$ such that the digraph becomes rooted k-edge-connected. (Frank showed in [4,6] that one can find such a v if the graph is k-partition-connected.) We will call this step of the algorithm an *elimination step*. If there is no appropriate v for a new edge, the elimination step fails and the algorithm returns a partition violating $e_G(\mathcal{P}) \geq k(|\mathcal{P}| - 1)$ that can be calculated in $O(n^4)$ running time (see [4]). ♣

We need to resolve two issues to reduce the running time. First, an elimination step is relatively slow since one needs to run a flow algorithm $O(n)$ times to determine whether there is any reversible path. Second, there are $O(nk)$ new edges resulting in $O(nk)$ elimination steps that need to run a flow algorithm $O(n^2k)$ times. To reduce the running time of the elimination steps, we present here a new simple data structure where k arc-disjoint one-way paths will be maintained from r_0 to v for every $v \in V - r_0$. Using these one-way paths it will be easy to check the reversibility of a path in the second phase of Algorithm 2.1 and it will be readily usable in a latter algorithm. Usually, these paths can be built up by running $n - 1$ flow algorithms; however in our cases the task will be simpler. Moreover, these paths can be updated easily when an arc is omitted or a path is reversed (after an omission of an arc), as follows. When a new arc r_0t is omitted, we omit the r_0v-path containing this arc from these k arc-disjoint paths if any exists and find k paths – if possible – using one augmenting path-searching step of the Ford–Fulkerson algorithm [3]. Note that if we update the data structure with this method, then there may remain only $k-1$ paths to some of the vertices and our algorithms will make a path reversal step to restore the rooted k-edge-connectivity. After a path P_0 is reversed, we first modify the P_0-arc-intersecting r_0v-paths, that is, the r_0v-paths intersecting P_0 in at least one arc. Where a path P enters P_0 at a point x we modify P such that from x we follow the reversed path $\overleftarrow{P_0}$ until another P_0-arc-intersecting r_0v-path P' leaves P_0 or we arrive at the start point of P_0. Thus we obtain at least $k-2$ one-way paths from r_0 to v (along with some circuits and some other paths) – as there were at least $k-1$ r_0v-paths before the reversal of P_0. With these $k-1$ paths we need only run one augmenting path-searching step of the Ford–Fulkerson algorithm to find k r_0v-paths. We call both methods – that is, both the one we perform after the omission of an arc and the one we perform after a path reversal – a *v-path update* or a *v-path check*, when we just want to check whether after reversing a path there remain k arc-disjoint r_0v-paths. One can see that the running time of a v-path update or check, respectively, is $O(n^2)$. We call the method when we call a v-path update for all $v \in V - r_0$ an *all-path update*.

Using this data structure we modify Algorithm 2.1 as follows.

Algorithm 2.2. Input: A graph $G = (V, E)$ and a root $r_0 \in V$.

Output: $\overrightarrow{G} = (V, \overrightarrow{E})$, an r_0-rooted k-edge-connected orientation of G along with the data structure of the k arc-disjoint r_0v-paths for every $v \in V - r_0$ OR a partition $\mathcal{P}$ of V with $e_G(\mathcal{P}) < k(|\mathcal{P}| - 1)$.

Phase 1: The same as *Phase* 1 of Algorithm 2.1.

Phase 2:

Step 1: Initialize the data structure of k one-way paths: take the k newly added arcs to every $v \in V - r_0$ as the k r_0v-paths. Label every node in $V - r_0$ with *non-inspected*.

Step 2: Let t be a non-inspected node.

Step 3: As long as there is a newly added r_0t-arc, omit it, and do an all-path update; and if there remain only $k-1$ arc-disjoint paths from r_0 to t, then for each $v \in V$ that is reachable from t in a path P_0, do a v-path check with the reversed path P_0 until we find a vertex to which there remain k arc-disjoint paths and we can finish with doing an all-path update for the current reversed path. If there is no appropriate v, then the elimination step fails and we can return a partition violating $e_G(\mathcal{P}) \geq k(|\mathcal{P}| - 1)$ as in Algorithm 2.1.

Step 4: Modify the label of t to *inspected*. If there is a non-inspected node go to *Step* 2; otherwise return the current orientation and the current state of the data structure. ♣

Algorithm 2.2 works since it is similar to Algorithm 2.1. It is easy to see that with the data structure, the running time of an elimination step is reduced from $O(n^4)$ (that is, the running time of $O(n)$ flow algorithms) to $O(n(n+m)) \leq O(n^3)$ (that is, the running time of $O(n)$ augmenting path-searching steps of the Ford–Fulkerson algorithm). The main point in *Step* 3 is that we omit the new arcs going to the same node sequentially; thus we can prove the following lemma.

Lemma 2.3. *If the omission of γ r_0t-edges for a vertex $t \in V - r_0$ is possible in* Step 3 *of Algorithm 2.2, then in these elimination steps we need to check the reversibility of $O(n+\gamma)$ paths starting at t. The reversibility of a tv-path can be checked by a v-path check; thus the omission can be done with $O(n+\gamma)$ path checks and $O(\gamma)$ all-path updates.*

Proof. Let T denote the set of vertices (currently) reachable from t. It is easy to see that after reversing a tv-path, no nodes in $V - T$ become reachable from t. However, T could become a smaller set.

After omitting one of these new edges, a path reversal is needed if there arises a set not containing r_0 with in-degree $k-1$. This set must contain t since the single omitted edge is r_0t and before its omission the digraph was rooted k-edge-connected. Hence by Menger's theorem, if there remain k arc-disjoint r_0t-paths, then the digraph remains rooted k-edge-connected. Thus after the omission we must do an all-path update and check whether there are still k arc-disjoint r_0t-paths. If there are not, then a path reversal is needed.

The reversal of a tv-path is sufficient to restore the rooted k-edge-connectivity if after its reversal the in-degrees of the $t\bar{r_0}$-sets become at least k and the in-degrees of the $v\bar{r_0}$-sets are not reduced under k since the in-degrees of the other subsets of $V - r_0$ do not change. Thus for a node v, there is a reversible path if $v \in T$, the in-degree of any $v\bar{r_0}$-set is at least k and the in-degree of any set containing v and not containing t and r_0 is at least $k+1$. Since the reversal of a path can increase the in-degree of any set by at most 1, if there was a $t\bar{r_0}$-set X with in-degree $k-1$ after removing some r_0t-edges and we restore its in-degree with a path reversal, then the omission of the next r_0t-edge reduces $\varrho(X)$ to $k-1$. The in-degree of a set not containing t and r_0 cannot increase and, as noted before, T becomes smaller and smaller. Therefore, if after the omission of some r_0t-edges there is no reversible tv-path, then it is not necessary to check v again. Thus every node v is checked at most once plus as many times as there have been tv-path reversals. Hence we need to check $O(n+\gamma)$ times. As we omitted γ edges and reversed at most γ paths, the number of all-path updates is clearly $O(\gamma)$. □

Therefore, we have the following corollary as we need to omit k r_0t-edges for each node $t \in V - r_0$.

Corollary 2.4. *The running time of Algorithm 2.2 is $O(n(n+k)n^2 + n^4) = O(n^4 + n^3k)$.* □

3. A simple algorithm for rooted $(k, 1)$-edge-connected orientability

In this section we give a new algorithmic proof for Theorem 1.1 when $\ell = 1$. The proof will be based on the following simple lemmas.

Let D/R (or G/R, respectively) denote the digraph (or graph, respectively) obtained by shrinking R into a single node r_R and deleting the loops while keeping the parallel edges. $D[R]$ denotes the digraph induced by R in D. For two disjoint subsets X and Y of the nodes of D, $\delta_D(X, Y)$ denotes the number of arcs with tail in X and head in Y.

Lemma 3.1. *Let $D = (V, A)$ be a digraph with a root node $r_0 \in V$. Let $R \subset V$ be a set of nodes containing r_0 for which $D[R]$ is r_0-rooted k-edge-connected and D/R is r_R-rooted k-edge-connected. Then D is r_0-rooted k-edge-connected.*

Proof. For a set $r_0 \in X \subset V$, if $R \not\subseteq X$, then $\delta_D(X) \geq \delta_D(X, R - X) \geq \delta_D(X \cap R, R - X) = \delta_{D[R]}(X \cap R) \geq k$ where the last inequality holds because of the r_0-rooted k-edge-connectivity of $D[R]$.

For a set $r_0 \in X \subset V$, if $R \subseteq X$, we get $\delta_D(X) = \delta_{D/R}(X - R + r_R) \geq k$ by the r_0-rooted k-edge-connectivity of D/R. $\square$

The next lemma holds for general ℓ, although we will need it only for $\ell = 1$.

Lemma 3.2. *Let $D = (V, A)$ be a digraph with a root $r_0 \in V$. Let $R \subset V$ be a set of nodes that contains r_0 and for which $D[R]$ is r_0-rooted (k, ℓ)-edge-connected and D/R is r_R-rooted (k, ℓ)-edge-connected. Then D is r_0-rooted (k, ℓ)-edge-connected.*

Proof. By using Lemma 3.1 both for D and for the reverse digraph $\overleftarrow{D}$ (for ℓ in place of k), we get that D is r_0-rooted k-edge-connected and $\overleftarrow{D}$ is r_0-rooted ℓ-edge-connected. Hence D is r_0-rooted (k, ℓ)-edge-connected. $\square$

In the special case $\ell = 1$, Theorem 1.1 is as follows.

Theorem 3.3. *A graph $G = (V, E)$ with a root node r_0 has an r_0-rooted $(k, 1)$-edge-connected orientation if and only if it is $(k, 1)$-partition-connected.*

Proof. As the necessity of $(k, 1)$-partition-connectivity is straightforward (after observing that in an r_0-rooted $(k, 1)$-edge-connected orientation of G, the in-degree of a member of a partition of V is at least 1 if it contains r_0, and at least k otherwise), we only prove sufficiency. We will use induction on $|V|$. Let $e_0 = r_0u \in E$ be an arbitrary edge. By the $(k, 1)$-partition-connectivity of G, $G - e_0$ is k-partition-connected. Hence $G - e_0$ has an r_0-rooted k-edge-connected orientation. This orientation gives us an orientation D of G if we orient e_0 towards r_0. Let R be the set of nodes in D from which there is a path to r_0. Thus $\varrho(R) = 0$ and $\varrho_{D[R]}(X) \geq 1$ whenever $r_0 \in X \subset R$. By the rooted k-edge-connectivity, there are k arc-disjoint paths from r_0 to v for every $v \in R - r_0$. Since $\varrho_D(R) = 0$, these paths cannot leave R. Therefore, by Menger's theorem, $\delta_{D[R]}(X) \geq k$ whenever $r_0 \in X \subset R$. Hence $D[R]$ is r_0-rooted $(k, 1)$-edge-connected. We also see that $|R| \geq 2$ because $r_0, u \in R$. If $R = V$, then we are done.

If $R \neq V$, we do the following. If $\mathcal{P} = \{X_1, X_2, \ldots, X_t\}$ is a partition of $V - R + r_R$, where $r_R \in X_1$, then $e_{\mathcal{P}}(G/R) = e_{\mathcal{P}'}(G) \geq k(|\mathcal{P}| - 1) + 1$ for the partition $\mathcal{P}' = \{X_1 - r_R \cup R, X_2, X_3, \ldots, X_t\}$ of V. Hence G/R is $(k, 1)$-partition-connected. By induction, there is an r_R-rooted $(k, 1)$-edge-connected orientation D' of G/R. Let $\overrightarrow{G}$ be the new orientation of G obtained by keeping the orientation of D on R and by orienting the other edges like in D'. Using Lemma 3.2 for $\overrightarrow{G}$ we get that $\overrightarrow{G}$ is r_0-rooted $(k, 1)$-edge-connected. $\square$

We note that this approach does not seem to work in the case where $\ell > 1$ since $D[R]$ need not be (k, ℓ)-edge-connected in this case. If with another definition we define $D[R]$ to be the maximal r_0-rooted (k, ℓ)-edge-connected subgraph of D, then the approach will also fail for $\ell > 1$ as $D[R]$ will be able to consist of the single node r_0. From the proof presented above, one can obtain the following algorithm.

Algorithm 3.4. Input: A graph $G = (V, E)$ and a root $r_0 \in V$.

Output: $\overrightarrow{G} = (V, \overrightarrow{E})$, an r_0-rooted $(k, 1)$-edge-connected orientation of G OR a partition $\mathcal{P}$ of V with $e_G(\mathcal{P}) < k(|\mathcal{P}| - 1) + 1$.

Let $e_0 \in E$ be an arbitrary edge adjacent to r_0. Run Algorithm 2.1 (or 2.2) on $G - e_0$ to decide whether there is an r_0-rooted k-edge-connected orientation of $G - e_0$. If no such orientation exists, the subroutine outputs a partition $\mathcal{P}$ with $e_G(\mathcal{P}) - 1 \leq e_{G-e_0}(\mathcal{P}) < k(|\mathcal{P}| - 1)$ and we return this partition.

Suppose now that this subroutine has found an orientation and let $D = (V, \overrightarrow{E}')$ be the digraph that we get by taking this orientation on $G - e_0$ and orienting e_0 towards r_0. We will denote the oriented mate of $e \in E$ in D with $\overrightarrow{e}'$.

Let R be the set of nodes from which r_0 is reachable in D that we could get by running any search algorithm. If $R = V$, then D is $(k, 1)$-edge-connected so we can return $\overrightarrow{G} := D$. Otherwise, for $e \in E[G]$, let $\overrightarrow{e} \in \overrightarrow{E}$ be $\overrightarrow{e}' \in \overrightarrow{E}'$ and run the algorithm recursively on G/R with root r_R (see Remark 3.5) and orient the edges not in $G[R]$ as this algorithm does. ♣

Remark 3.5. In Algorithm 3.4, $|R| > 1$ because r and the other endpoint of e_0 are in R. Hence we can run the algorithm on G/R recursively.

It is easy to see that if the subroutine outputs a partition $\mathcal{P}'$ of the node-set of the possibly contracted graph G' with $e_{G'-e_0}(\mathcal{P}') < k(|\mathcal{P}'| - 1)$, then we can modify it easily to get a partition $\mathcal{P}$ of V with $e_G(\mathcal{P}) < k(|\mathcal{P}| - 1) + 1$. Namely, let $\mathcal{P}$ be the partition that we get by changing the new node r^* that represents the contracted set to the set that it represents in the member of $\mathcal{P}'$ containing r^*.

One can see that the algorithm runs the subroutine and the search algorithm $O(n)$ times; hence the running time of the algorithm is $O(n(\vartheta + n + m))$ where ϑ is the running time of the subroutine.

3.1. Extension to hypergraphs

To extend the algorithm to hypergraphs we need to consider directed hypergraphs. A directed hypergraph or *dypergraph* $\mathcal{D} = (V, \mathcal{A})$ consists of the node-set V and the set $\mathcal{A} \subseteq 2^V$ of directed hyperedges. Here, as in [7], a directed hyperedge, called a *dyperedge*, has one head node while all of its other nodes are the tails. (We assume that a dyperedge of a dypergraph and a hyperedge of a hypergraph consist of at least two nodes.)

One can define (k, ℓ)-partition-connectivity of hypergraphs and rooted (k, ℓ)-edge-connectivity of dypergraphs like for graphs (see [7]). We note that Menger's theorem can be extended to dypergraphs by using Menger's theorem for the digraph that we get by changing every dyperedge to a new node and arcs from every tail of the dyperedge to this node and one arc from the new node to the head of the dyperedge. Frank, Király and Király [7] extended Theorem 1.1 to hypergraphs with an algorithmic proof. The case where $\ell = 0$ can be solved by using Edmonds' matroid partition algorithm [2]. To extend Algorithm 3.4, one needs this algorithm as a subroutine. Observe that the proof of the lemmas and Theorem 3.3 is nearly the same as for graphs. The single issue is the following. In the proof of the extension of Lemma 3.2, we cannot use the extended Lemma 3.1 for the reverse dypergraph as it cannot be well defined. Hence we need to prove a hypergraphic counterpart of Lemma 3.1 where rooted $(0, \ell)$-edge-connectivity is considered. Fortunately, the same proof works; one only needs to substitute δ with ϱ in the proof. Therefore, the extension of Algorithm 3.4 works well for hypergraphs.

4. A quicker algorithm for rooted $(k, 1)$-edge-connected orientability

In this section we modify Algorithm 3.4 to achieve a running time of $O(n^4 + n^3k)$ for graphs. The main idea of this modification is the following. Instead of reorienting every edge of G/R in each step, we keep the orientation given by D/R and augment it using the idea of *Step* 3 of Algorithm 2.2. As in Section 2 for every $v \in V - r_0$, k arc-disjoint one-way paths will be stored from r_0 to v. It is easy to see that these paths give the same structure on D/R for a set R with $r_0 \in R \subseteq V$ if we cut down the first part of them from r_0 to their last node in R. Now we are ready to describe the algorithm.

Algorithm 4.1. INPUT: A graph $G = (V, E)$; and a root $r_0 \in V$.

OUTPUT: $\overrightarrow{G} = (V, \overrightarrow{E})$, an r_0-rooted $(k, 1)$-edge-connected orientation of G OR a partition $\mathcal{P}$ of V with $e_G(\mathcal{P}) < k(|\mathcal{P}| - 1) + 1$.

Step 1: Let $e_0 = r_0u \in E$ be an arbitrary edge. Decide whether there is an r_0-rooted k-edge-connected orientation of $G - e_0$ with Algorithm 2.2. If no such orientation exists, the subroutine outputs a partition $\mathcal{P}$ with $e_G(\mathcal{P}) - 1 \le e_{G-e_0}(\mathcal{P}) < k(|\mathcal{P}| - 1)$ and we return this partition.

Step 2: Suppose now that Algorithm 2.2 has found an orientation and let $D = (V, \overrightarrow{E}')$ be the digraph that we get by taking this orientation on $G - e_0$ and orienting e_0 towards r_0. We will denote the directed pair of $e \in E$ in D with $\overrightarrow{e}$. Note that the k arc-disjoint r_0v-paths for every $v \in V - r_0$ given by Algorithm 2.2 for the orientation of $G - e_0$ are still present in D.

Step 3: Run any search algorithm to find the set of nodes R from which r_0 is reachable in D. If $R = V$, then D is $(k, 1)$-edge-connected, so we can return $\overrightarrow{G} := D$. Otherwise, go to *Step* 4.

Step 4: Let $\overrightarrow{e}$ be an arc of D/R leaving r_R. Run an elimination step for $\overrightarrow{e}$ as in *Step* 3 of Algorithm 2.2. If the elimination step fails, output the partition that is given by *Step* 3 of Algorithm 2.2 (after substituting r_R with all the nodes in R in the member of the partition containing r_R). Otherwise, update D to this new graph along with adding the reversed pair $\overleftarrow{e}$ of $\overrightarrow{e}$ to it and go to *Step* 3. ♣

It is easy to see that Algorithm 4.1 works, as it is just a modification of Algorithm 3.4. Hence we only prove that its running time is $O(n^4 + n^3k)$.

Theorem 4.2. *Algorithm 4.1 runs in time $O(n^4 + n^3k)$.*

Proof. As we have seen in Corollary 2.4, *Step* 1–2 runs in time $O(n^4 + n^3k)$. *Step* 3–4 runs $O(n)$ times since $|R|$ increases in each run of *Step* 3. In *Step* 3 we run a search algorithm; hence the running time of this step is $O(n^2)$. By Lemma 2.3 the running time of *Step* 4 is $O(n^3)$ if we find an augmenting path and $O(n^4)$ otherwise, but this case could only happen once, when the algorithm terminates by outputting a partition. Thus the total running time of *Step* 3–4 in the whole algorithm is $O(n^4)$. Therefore, the algorithm runs in time $O(n^4 + n^3k)$. □

Acknowledgments

The author received grants (No. CK 80124 and No. K 109240) from the National Development Agency of Hungary, based on a source from the Research and Technology Innovation Fund. The research was supported by the MTA-ELTE Egerváry Research Group.

The author is grateful to András Frank for inspiring discussions and his comments.

References

[1] R. Connelly, T. Jordán, W. Whiteley, Generic global rigidity of body–bar frameworks, J. Combin. Theory Ser. B (2013) in press (http://dx.doi.org/10.1016/j.jctb.2013.09.002). See also Technical Report TR-2009-13, Egerváry Research Group, Budapest, 2009. www.cs.elte.hu/egres.
[2] J. Edmonds, Minimum partition of a matroid into independent sets, J. Res. Natl. Bur. Stand., Sect. B 69 (1965) 67–72.
[3] L.R. Ford, D.R. Fulkerson, Flows in Networks, Princeton Univ. Press, Princeton, 1962.
[4] A. Frank, On disjoint trees and arborescences, in: Algebraic Methods in Graph Theory, in: Colloquia Mathematica Soc. J. Bolyai, vol. 25, North-Holland, 1978, pp. 59–169.
[5] A. Frank, On the orientation of graphs, J. Combin. Theory Ser. B 28 (3) (1980) 251–261.
[6] A. Frank, Connections in Combinatorial Optimization, Oxford Univ. Press, 2011.
[7] A. Frank, T. Király, Z. Király, On the orientation of graphs and hypergraphs, Discrete Appl. Math. 131 (2) (2003) 385–400.
[8] A. Frank, É. Tardos, Generalized polymatroids and submodular flows, Math. Program. 42 (1988) 489–563.
[9] Harold N. Gabow, K.S. Manu, Packing algorithms for arborescences (and spanning trees) in capacitated graphs, Math. Program. 82 (1998) 83–109.
[10] S.L. Hakimi, On the degrees of the vertices of a directed graph, J. Franklin Inst. 279 (4) (1969) 290–308.
[11] Cs. Király, Algoritmusok szupermoduláris függvények fedésére, Master's Thesis, 2010 (in Hungarian).
[12] T. Naitoh, S. Fujishige, A note on the Frank–Tardos bi-truncation algorithm for crossing-submodular functions, Math. Program. 53 (1992) 361–363.
[13] W.T. Tutte, On the problem of decomposing a graph into n connected factors, J. Lond. Math. Soc. 142 (1961) 221–230.