Zuse Institute Berlin

Takustr. 7
14195 Berlin
Germany

KAI HELGE BECKER, BENJAMIN HILLER

Efficient Enumeration of Acyclic Graph Orientations with Sources or Sinks Revisited

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – The Berlin Mathematics Research Center MATH+ (EXC-2046/1, project ID: 390685689). Moreover, the authors thank the BMBF Research Campus Modal (fund number 05M14ZAM) for additional support.

ZIB Report 20-05 (February 2020)

Zuse Institute Berlin
Takustr. 7
14195 Berlin
Germany

Telephone: +49 30-84185-0
Telefax: +49 30-84185-125

E-mail: `bibliothek@zib.de`
URL: `http://www.zib.de`

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

Efficient Enumeration of Acyclic Graph Orientations with Sources or Sinks Revisited

Kai Helge Becker and Benjamin Hiller

March 1, 2020

Abstract

In a recent paper, Conte et al. [1] presented an algorithm for enumerating all acyclic orientations of a graph $G = (V, E)$ with a single source (and related orientations) with delay $\mathcal{O}(|V||E|)$. In this paper we revisit the problem by going back to an early paper by de Fraysseix et al. [12], who proposed an algorithm for enumerating all bipolar orientations of a graph based on a recursion formula. We first formalize de Fraysseix et al.'s algorithm for bipolar orientations and determine that its delay is also $\mathcal{O}(|V||E|)$. We then apply their recursion formula to the case of Conte et al.'s enumeration problem and show that this yields a more efficient enumeration algorithm with delay $\mathcal{O}(\sqrt{|V||E|})$. Finally, a way to further streamline the algorithm that leads to a particularly simple implementation is suggested.

1 Introduction

Graph orientations and their characteristics have long been a subject of mathematical inquiry. A special case are bipolar orientations (or s-t-orientations), i.e., acyclic orientations with two distinguished nodes s and t such that s is the only node without outgoing arcs (called the source) and t is the only node without incoming arcs (called the sink). The first to define the concept were [2] in the context of a planarity test, who also proved that an undirected graph $G = (V, E)$ has a bipolar orientation if and only if $G' = (V, E \cup \{s, t\})$ is 2-connected. Even, Tarjan and Ebert [9, 8, 19] developed linear-time algorithms for constructing a bipolar orientation of a given graph. In [12] de Fraysseix, de Mendez and Rosenstiehl presented a comprehensive survey of properties of bipolar orientations found up to then and a number of new properties. In particular, they provided a recursion formula for bipolar graphs and on this basis sketched an algorithm that allows for enumerating all bipolar orientations of a given graph. Bipolar orientations have found many applications, among these in lattice theory [14] and graph drawing [15, 13].

A more general case of graph orientations are acyclic orientations with a unique source and no restriction on the number of sinks. While these single-source acyclic orientations have not been studied as extensively, they are related to Whitney numbers [11], to the chip-firing game [4], and to maximum parking functions and spanning trees [5], for example. Recently, Conte et al. [1] have provided an algorithm to enumerate all single-source acyclic orientations of a given graph and related multiple-source acyclic orientations with delay $\mathcal{O}(|E||V|)$.

In this paper we revisit the problem of enumerating single-source acyclic orientations (and related acyclic orientations) on the basis of the work by de Fraysseix, de Mendes and Rosienstiehl [12]. In the next section we will formalize the algorithm by de Fraisseix et al. and determine its delay between two bipolar orientations. In Section 3 we apply their recursion formula for bipolar orientations to the case of single-source acyclic orientations and show that in this case, de Fraysseix's et al.'s algorithm leads to a delay of $\mathcal{O}(|E|\sqrt{|V|})$ and thus improves the result in [1]. Moreover, we discuss a leaner implementation of their algorithm for the case of single-source acyclic orientations. We conclude in Section 4 with a some suggestions for further research.

We end this introductory section with some terminological remarks. Throughout this paper we will consider multigraphs $G = (V, E)$, i.e. graphs that may have multiple edges or loops. As we do not have to distinguish between different edges between the same pair of nodes, it is sufficient to consider E a multiset in the following. As usual, $deg(u)$ refers to the degree of a node u, n to the number of nodes of G and m to the number of its edges, and to exclude trivial cases we will assume $m \geq 2$ in this paper. If we delete an edge $e \in E$ of G, the resulting graph will be referred to as $G \setminus e$, and when an edge is contracted (i.e. deleted with the ends being identified) the resulting graph is denoted by G/e. A graph is 2-connected if it is still connected after the removal of one vertex. In the context of multigraphs it is common to understand the removal of a vertex with a loop as separating the loop from the rest of the graph. Accordingly, 2-connected multigraphs are always loopless [20, 7]. (Note that this implies no restriction for our topic since loops prevent a graph from having an acyclic orientation.)

Finally, for a given graph G, the number of bipolar orientations of G is denoted by $\theta_{s,t}(G)$, where s is the source and t the sink, and the number of single-source acyclic orientations by $\zeta_s(G)$, where s is the source.

2 Enumerating bipolar orientations

We will now revisit the algorithm for bipolar orientations in [12] and begin with a well-known result [2].

Proposition 1 *A multigraph $G = (V, E)$ with source $s \in V$ and sink $t \in V$ admits an s-t-orientation if and only if $G' = (V, E \cup \{s, t\})$ is 2-connected.* □

The starting point for the algorithm is the following statement in [12]:

Proposition 2 *For a 2-connected multigraph $G = (V, E)$ with source s and sink t, the following recursive equation holds:*

$$\theta_{s,t}(G) = \theta_{s,t}(G \setminus e) + \theta_{s,t}(G/e)$$

for all $e \in E \setminus \{s, t\}$. □

The recursion formula suggests the following algorithm already sketched in [12].

Begin with an edge e that the source is incident with and create the graphs G/e and $G \setminus e$. Provided that the resulting graph is still 2-connected (otherwise $\theta_{s,t}(G/e) = 0$ or $\theta_{s,t}(G \setminus e) = 0$), continue creating the graphs G/e and $G \setminus e$ by recursively choosing an edge that the source is incident with, until the remaining graph has node set $\{s, t\}$ and edge set $\{\{s, t\}\}$. (Note that Proposition 2 implies that at least one of the two operations "contraction" and "deletion" will always lead to a 2-connected graph.)

This results in a binary tree the nodes of which represent graphs, where each leaf of the tree represents the graph with node set $\{s, t\}$ and edge set $\{\{s, t\}\}$, and each path from the root node (the original graph G) to a leaf represents one bipolar orientation of G.

This orientation can be constructed as follows: Replace the edge $\{s, t\}$ by the arc (s, t). Now go backwards along the path from the leaf of the tree to its root. At each node of the path reverse the contraction or deletion operation that led to the current node and insert the contracted or deleted edge as an arc outgoing from the source. Once the root node has been reached, all edges of the original graph have been oriented to yield a bipolar orientation.

Our following Algorithm 1 formalizes this algorithmic idea in detail. Here the list L keeps track of all nodes and edges that were removed during the recursive procedure of contracting and removing edges, while the list $\mathcal{D}$ contains all bipolar orientations of the input graph when the algorithm terminates. Note that in Algorithm 1 a contraction is attempted before a deletion because this way of generating the binary tree turned out to be computationally advantageous.

In the following we will analyse the computational time Algorithm 1 requires to enumerate all bipolar orientations. We will begin with a lemma that counts the

Algorithm 1: Enumerating Bipolar Orientations

```
Input   : A 2-connected multigraph $G = (V, E)$ with source $t \in V$,
          sink $t \in V$, and $\{s, t\} \notin E$
. Output : A list $\mathcal{D}$ of all bipolar orientations $D = (V, A)$ of $G$
$\mathcal{D} := [..]$
$L := [..]$ % Doubly-linked list or array to keep track of graph changes
$E := E + \{s, t\}$
AnalyseGraph $((V, E), L, \mathcal{D})$
Function AnalyseGraph($(V, E), L, \mathcal{D}$)
    if $|E| > 1$ then
        % Pick edge for contraction or deletion
        Pick $\{s, v\} \in E \setminus \{s, t\}$
        % Is edge contraction possible?
        $E^- := \{\{v, w\} \in E \mid w \in N(v) - \{s\}\}$
        $E^+ := \{\{s, w\} \mid w \in N(v) - \{s\}\}$
        $E := E - \{\{s, v\}\} - E^- + E^+$
        $contraction := FALSE$
        if $(V, E)$ is 2-connected then
            % Contract edge
            $contraction := TRUE$
            add $(v, E^-, E^+)$ to list $L$
            AnalyseGraph $((V, E), L, \mathcal{D}))$
            remove $(v, E^-, E^+)$ from list $L$
        $E := E + \{\{s, v\}\} + E^- - E^+$
        % Is edge deletion possible?
        $E := E - \{\{s, v\}\}$
        if (contraction $= FALSE \vee (V, E)$ is 2-connected) then
            % Delete edge
            add $(v, \varnothing, \varnothing)$ to list $L$
            AnalyseGraph $((V, E), L, \mathcal{D}))$
            remove $(v, \varnothing, \varnothing)$ from list $L$
        $E := E + \{\{s, v\}\}$
    else
        % Generate orientation
        % Proceed in L from last to first entry in the following
        for $(v, E^-, E^+) \in L$ do
            $A := A + \{(s, v)\} + \{(v, w) : \{v, w\} \in E^-\}$
                    $-\{(s, w) : \{s, w\} \in E^+\}$
        add $(V, A)$ to $\mathcal{D}$
    return $(V, E), L, \mathcal{D}$
```

number of edge contractions and deletions from the root to a leaf of the graph analysis tree.

Lemma 1 *In Algorithm 1, each leaf of the binary tree that is built when analysing the graph, is reached by $n-2$ edge contractions and $m-n+2$ edge deletions from the root of the tree.* □

Proof A leaf of the binary tree in Algorithm 1 has been reached when the graph G has been reduced, by a series of deletions and contractions, to the graph with node set $\{s,t\}$ and edge set $\{\{s,t\}\}$. For doing so, $n-2$ nodes have to be removed, i.e., $n-2$ edge contractions have to take place. Moreover, as altogether m edges have to be removed, we require $m-(n-2)$ edge deletions. ■

Proposition 3 *Let $G=(V,E)$ be a multigraph with source $s \in V$ and sink $t \in V$ and with $\{s,t\} \notin E$ such that $G' = (V, E \cup \{s,t\})$ is 2-connected. Assume that we can maintain a data structure for checking whether a graph is 2-connected with update time $g(n,m)$ after edge contraction or edge deletion and with query time $q(n,m)$. Further let σ be the total number of edges to be carried over from v to s when trying an edge contraction. Then Algorithm 1 enumerates all bipolar orientations with delay at most $(4\sigma+3m)g(m,n)+(m+n-2)q(m,n)+15m-n-4deg(s)+2$, where $\sigma \leq 2m$.* □

Proof When constructing an (s,t)-bipolar orientation with Algorithm 1, the worst case occurs when the algorithm begins with an edge that can be both contracted and deleted, such that at some point of the algorithm we have to go from a leaf of the tree all the way up to the root and from there down to the leaf. When we backtrack from a leaf to the root of the tree, each node will have been reached by an edge contraction or deletion from its parent node.

Carrying out a contraction requires us, apart from removing a node, to carry over the edges between the node to be contracted and its neighbour to edges between the supersink and the neighbours of the node to be contracted. In Algorithm 1 edges are carried over before checking whether a contraction is possible (line 13), and after a contraction or an unsuccessful check for contraction this operation is reversed to recover the previous graph (line 21). On the path up to the root we only carry out the latter procedure, after alltogether $n-2$ operations of removing (v, E^+, E^-) from the list. As each of the m edges to be deleted or contracted during the course of the algorithm has to be at the source s for a deletion or contraction to occur, the algorithm carries over altogether $m - deg(s)$ edges during it path from the root to a leaf, where carrying over one edge requires one operation to delete the edge at v and one operation to insert the edge at s. As a consequence, recovering the graph prior to contraction requires a total $2(m-deg(s))g(n,m)$ time for carrying over edges after contraction, when we take into account the time $g(m,n)$ to maintain our data structure. As altogether $(n-2)$ edges are contracted during the course of the algorithm according to Lemma 1, we need another $(n-2)g(m,n)$ time for adding the contracted edges. After recovering the previous graph, Algorithm 1 checks whether an edge deletion is possible. Deleting an edge, checking the contraction flag, testing for 2-connectivity and adding the edge to recover the previous graph takes altogether $(n-2)(2g(n,m)+q(n,m))+n-2$ time, including the time to maintain the data structure and the query for 2-connectedness. (Note that in the case considered here, i.e., where we go up the entire path to the root of the tree, it will always turn out that a deletion is not possible and we will backtrack one node further up.) If instead of a contraction a deletion has taken place, all we have to do is to add an edge to recover the graph prior to deletion. As this happens $m - n + 2$ times according to Lemma 1, we require altogether $(m - n + 2)g(n, m) + m - n + 2$ time for these operations, including closing the subroutine $m - n + 2$ times.

To conclude, backtracking from a leaf of the tree to the root can be accomplished with altogether $(3m + 2n - 2deg(s) - 4)g(n, m) + (n - 2)q(n, m) + m + n - 2$ operations.

Going down from the root to a leaf involves m steps, at each of which the algorithm checks whether $|E| > 1$ and picks and edge to try an edge contraction, which requires $2m$ operations. For the contraction, altogether σ edges will have to be carried over, which results in $2\sigma g(n, m)$ operations, and m edges are deleted ($mg(m, n)$ operations). Setting the contraction flag to $False$ involves m steps, while checking whether an edge contraction is possible because the resulting graph is 2-connected takes $mq(n, m) + m$ operations. In $n - 2$ cases (Lemma 1) the edge contraction will turn out to be feasible and the algorithm will need $n-2$ operations to set the contraction flag to $True$ and $2(n - 2) + 2(m - deg(s))$ to add the $n - 2$ nodes and the $2(m - deg(s))$ edges to L and call the function. In those $m - n + 2$ cases (Lemma 1) where the contraction turned out to be infeasible, the algorithm proceeds to carrying out a deletion instead. Recovering the graph prior to contraction takes altogether $(m-n+2)g(n, m)+2\sigma g(m, n)-2(m-deg(s))g(m, n)$ operations (note that $2(m - deg(s))g(m, n)$ operations take place when the contraction was successful). Finally, the deletion of the $m - n + 2$ edges that follows requires $(m - n + 2)g$ operations, checking that the contraction flag is $False$ indeed $m - n + 2$ operations and adding the required information to L $3(m - n + 2)$ operations, and calling the function another $(m - n + 2)$ operations.

Hence, the path from the root to the leaf will take a total of $(4\sigma - 2n + 2deg(s) + 4)g(m, n) + mq(m, n) + 11m - 2n - 2deg(s) + 4$ operations.

Finally we have to consider the time it takes to generate an orientation. For generating an orientation, altogether m arcs have to be inserted. Moreover, as we generate an orientation by reverting the process of contractions and deletions that took us from the root of the tree to the relevant leaf, we have to carry over $m - deg(s)$ arcs, which requires $2(m - deg(s))$ operations, i.e., we arrive at a total time of $3m - 2deg(s)$ for generating an orientation.

Summing up the number of operations required for going up from a leaf to the root and down to another leaf and the time of generating the orientation yields the required result.

Finally, every edge $\{u, v\}$ can be carried over at most twice in an attempt to find a feasible contraction, namely when the edge $\{s, u\}$ or the edge $\{s, v\}$ is to be contracted. Hence, $2m$ is an upper bound on σ. ■

For the general case of testing whether a graph is 2-connected, [18] and [16] have provided $\mathcal{O}(m)$ algorithms. Using the sparsification technique in [6], computational time can be improved to $\mathcal{O}(n)$ per edge addition, deletion and query, which yields the following corollary of Proposition 3 with $g(m, n) = q(m, n) = n$.

Corollary 1 *Let $G = (V, E)$ be a 2-connected multigraph with supersource $s \in V$ and supersink $t \in V$. Algorithm 1 can enumerate all bipolar orientations with delay $\mathcal{O}(mn)$.* □

3 Enumerating single-source acyclic orientations

We will now turn to the problem of enumerating all single-source acyclic orientations of a graph as addressed by Conte, Grossi, Marino and Rizzi [1].

The following proposition shows that this problem can directly be reduced to the problem of finding all bipolar orientations of a graph. As a consequence, we will arrive at a more efficient algorithm for the enumeration of single-source acyclic orientations.

Proposition 4 *Let $G = (V, E)$ be a multigraph with source $s \in V$.*
(i) G has a single-source acyclic orientation if and only if it is connected and loopless.
(ii) If G is connected and $|E| \geq 2$, then $\zeta_s(G)$ satisfies the following recursive equation:

$$\zeta_s(G) = \zeta_s(G \setminus e) + \zeta_s(G/e)$$

for all $e \in E$ that are not loops. □

Proof We create a graph $G' = (V', E')$ by adding a node t (the supersink) to V and add edges connecting t to every node, including the supersource s, i.e.,

$$V' = V \cup \{t\} \text{ and } E' = E \cup \{\{u, t\} \mid u \in V\}.$$

Now each (s, t)-bipolar orientation of G' corresponds to an acyclic orientation with supersource s on G, and vice versa.
(i) If we show that G' is 2-connected if and only if $G' \setminus \{t\} = G$ is connected and loopless, Proposition 1 finishes this part of the proof. Trivially, if G' is 2-connected, G is connected and loopless. Conversely, let G be connected and loopless and $u, v \in V' \setminus \{t\}$, with $u \neq v$. Then there are two node-disjoint paths between u and v on G': the path $u - t - v$ and a path with edges from E since G is connected. Moreover, there are two node-disjoint paths between t and u: the path $u - t$ and a path that consists of both the edge $\{t, s\}$ and a path from s to V with edges from E.
(ii) We have

$$\zeta_s(G) = \theta_{s,t}(G') = \theta_{s,t}(G' \setminus e) + \theta_{s,t}(G'/e) = \zeta_s(G \setminus e) + \zeta_s(G/e),$$

where the first and the last equalities hold since the arcs incident to t all have t as its head and hence have no impact on the number of orientations, and the second equality holds due to Proposition 2 and Proposition 1 because G' is 2-connected if G is connected, as we have seen in part (i) of this proof. ■

The previous proposition implies that we can use Algorithm 1 straight away to enumerate all single-source acyclic orientations for a given graph G, with the only (welcome) modification being that we have to check for connectedness instead of 2-connectedness when attempting a contraction or a deletion of an edge.

Connectedness, however, can be checked easily in $\mathcal{O}(m)$ time via depth-first search. Using a more sophisticated algorithm developed in [10], building on previous work in [17], a data structure for testing connectedness can be maintained in $\mathcal{O}(\sqrt{m})$ time with query time $\mathcal{O}(1)$. The general idea here is to assign a weight of 1 to all edges of the graph and a weight of 2 to all possible other edges. The data structure in question maintains a minimum spanning tree of the overall graph, the weight of which reveals whether the graph is connected. Using the sparsification technique in [6] this result can be improved to a maintenance time of $g(n, m) = \mathcal{O}(\sqrt{n})$ and a query time of $q(n, m) = \mathcal{O}(1)$.

The following corollary of Proposition 3 shows that Algorithm 1 when modified for the case of single-source acyclic orientations is computationally less expensive with respect to delay than the algorithm in [1].

Corollary 2 *For a connected loopless multigraph $G = (V, E)$ with a single supersource $s \in V$, Algorithm 1 enumerates all acyclic orientations with delay $\mathcal{O}(m\sqrt{n})$.* □

Proof Directly follows from Proposition 3 and Proposition 4 with $g(n, m) = \mathcal{O}(\sqrt{n})$ and $q(n, m) = \mathcal{O}(1)$. ■

In the remainder of this section we show how Algorithm 1 can be further improved. While this does not allow us to reduce the complexity below $\mathcal{O}(\sqrt{n}m)$, it significantly speeds up our algorithm.

We recall that Algorithm 1 has to check whether the graph that results from contracting an edge is 2-connected (or connected, in the case of generating single-source acyclic orientations) and loopless and, for carrying out this check, the algorithm has to create this graph first. In the present case of generating all single-source acyclic orientations, however, we can easily know beforehand whether the resulting graph is connected and loopless.

Lemma 2 *A connected loopless multigraph $G = (V, E)$ with supersource $s \in V$ is connected and loopless after contracting $e := \{s, u\}$ if and only if s and u are joined by exactly one edge prior to the contraction.* □

Proof As contracting an edge does not affect the connectivity of a graph, the only criterion to check is looplessness. Clearly, the graph is loopless if and only if s and u are joined by exactly one edge prior to the contraction. ■

For improving our algorithm, this implies that we do not have to carry over edges to the source s before knowing whether the contraction will be successful and do not have to spend computation effort on recovering the original graph when the contraction turned out to be unsuccessful. Instead of these m attempts at contracting an edge, we can restrict ourselves to carrying out the $n - 1$ successful contractions (cf. Lemma 1). All we have to do against the background of Lemma 2 is to maintain an array $k[..]$ that stores the number of edges that connect the source s with all other nodes of the graph. If and only if $k[u] = 1$ for a particular node u we can carry out the contraction.

The new situation is given by Algorithm 2, where carrying over edges to the source s takes place after the algorithm has decided to contract the current edge (lines 15 to 17). The consequences for the delay are given in the following proposition.

Proposition 5 *For the case of generating all single-source acyclic orientations, the delay of Algorithm 2 is*

$$(4\sigma - 3m - 2n + 4deg(s) + 4)g(m,n) + mq(m,n) + 3m + n + deg(s) - 2$$

steps of computational time shorter than the one of Algorithm 1. □

Proof Algorithm 2 differs from Algorithm 1 only with respect to the path from the root down to a leaf. Here, we need $4m$ steps until we have picked an edge, set the contraction flag to $False$ and checked whether it is not a multi-edge. If a contraction is possible, which is the case $n-2$ times according to Lemma 1, we require altogether $n-2$ steps to set the contraction flag to $True$, altogether $2(m - deg(s))g(m,n)$ operations to carry over the relevant edges to s, altogether $(n-2)g(m,n)$ operations to delete the contracted edges, altogether $m - deg(s)$ operations to update the array $k[..]$, have to add altogether $2(m - deg(s))$ edges and $n-2$ nodes to the data structure L that keeps track of graph changes, and call the function $n-2$ times.
In the cases where a contraction has not been not possible, which occurs in $m-n+2$ cases according to Lemma 1, there are altogether $m-n+2$ checks whether the contraction flag is set to $False$, deleting edges takes altogether $(m-n+2)g(m,n)$ time, updating the array $k[..]$ requires $m-n+2$ operations, keeping book of the operation in our data structure L leads to a total of $3(m-n+2)$ operations, and calling the function requires $m-n+2$ steps.
All in all, going down the path from the root to a leaf requires

$$(3m - 2deg(s))g(m,n) + 8m - 3n - 3deg(s) + 6$$

operations. Comparing this with the

$$(4\sigma - 2n + 2deg(s) + 4)g(m,n) + mq(m,n) + 11m - 2n - 2deg(s) + 4$$

operations that Algorithm 1 needs according to the proof of Proposition 3 yields the result. ■

So far we have discussed computational time as a function of the time $q(m,n)$ it takes to query whether the graph after a contraction or deletion is still (2-)connected, and as a function of the time $g(m,n)$ it takes to update a data structure that enables us to perform such a query efficiently. In the case of Algorithm 1 such a query was required in each case of trying out a contraction and in each case of a deletion when a contraction was carried out right before. (Recall that when no contraction was carried out, a deletion is always possible.)

In Algorithm 2, however, we do not need to check connectedness when trying out a contraction, i.e., the only case where a query with time $q(m,n)$ occurs is in those $n-2$ cases (see Lemma 1) when we are at an edge that was contracted before and we have to check whether a deletion is feasible, too. This suggests that there may be practical applications for which maintaining a specialised data structure for checking connectedness efficiently is not necessary.

Algorithm 2: Enumerating Single-source Acyclic Orientations

```
Input  : A loopless connected multigraph $G = (V, E)$ with source
         $s \in V$
Output : A list $\mathcal{D}$ of all acyclic orientations $D = (V, A)$ of $G$
$\mathcal{D} := [..]$
$L := [..]$ % Doubly-linked list or array to keep track of graph changes
% Initiate array to keep track of multi-edges incident to source
for $u \in N(s)$ do k[$u$ ]:= Number of edges between $u$ and $s$
% Start graph analysis
AnalyseGraph $((V, E), L, \mathcal{D})$
Function AnalyseGraph($(V, E), L, \mathcal{D}$)
    if $E \neq \varnothing$ then
        % Pick edge for contraction or deletion
        Pick $\{s, v\} \in E$
        % Is edge contraction possible?
        $contraction := FALSE$
        if $k[v] = 1$ then
            $contraction := TRUE$
            $E^- := \{\{v, w\} \in E \mid w \in N(v) - \{s\}\}$
            $E^+ := \{\{s, w\} \mid w \in N(v) - \{s\}\}$
            $E := E - \{\{s, v\}\} - E^- + E^+$
            for $\{s, w\} \in E^+$ do $k[w] := k[w] + 1$
            add $(v, E^-, E^+)$ to list $L$
            AnalyseGraph $((V, E), L, \mathcal{D}))$
            remove $(v, E^-, E^+)$ from list $L$
              $E := E + \{\{s, v\}\} + E^- - E^+$
        $E := E - \{\{s, v\}\}$
        % Is edge deletion possible?
        if (contraction $= FALSE \vee (V, E)$ is connected) then
            $k[v] := k[v] - 1$
            add $(v, \varnothing, \varnothing)$ to list $L$
            AnalyseGraph $((V, E), L, \mathcal{D}))$
            remove $(v, \varnothing, \varnothing)$ from list $L$
        $E := E + \{\{s, v\}\}$
    else
        % Generate orientation
        % Proceed in L from last to first entry in the following
        for $(v, E^-, E^+) \in L$ do
            $A := A + \{(s, v)\} + \{(v, w) : \{v, w\} \in E^-\}$
                    $-\{(s, w) : \{s, w\} \in E^+\}$
        add $(V, A)$ to list $\mathcal{D}$
    return $(V, E), L, \mathcal{D}$
```

The previous proposition implies that for the case of single-source acyclic orientations, we save a total of $\mathcal{O}(m\sqrt{n})$ delay with Algorithm 2 compared with Algorithm 1 when we choose the best known implementation with $g(m,n) = \mathcal{O}(()\sqrt{n})$ and $q(m,n) = \mathcal{O}(1)$. Unfortunately, while this improvement is certainly not negligible from a practical perspective, it does not improve the overall complexity of Algorithm 2 beyond $\mathcal{O}(m\sqrt{n})$ because the most expensive operations are related to maintaining the specialized data stucture necessary for the query whether the graph is connected after deleting and edge that was contracted before.

Doing without such a data structure and using a standard depth-first-search algorithm for checking connectedness implies $g(m,n) = 1$ and $q(m,n) = \mathcal{O}(m)$. This leads to an $\mathcal{O}(m)$-delay while going down the tree from the root to a leaf and an overall $\mathcal{O}(mn)$-delay for generating single-source acyclic orientations, i.e., to an algorithm with the same complexity as the algorithm proposed by Conte et al. [1], albeit with a much simpler implementation than their approach.

Finally, let us mention that our Algorithm 2 for single-source acyclic orientations can also be applied to some related problems addressed by Conte et al., namely the problems that they call "weak single-source acyclic orientations", "weak multiple-source acyclic orientations", and "strong multiple-source acyclic orientations". Since these problems, as Conte et al. have observed, can be transformed into the problem of generating single-source acyclic orientations in $\mathcal{O}(m)$ time, they can be solved with the same delay as the problem we have discussed in the present section.

4 Conclusion

In this paper we have formalized and determined the delay of an algorithm for generating all bipolar orientations of a graph that was proposed de Fraysseix et al. [12] on the basis of a recursion formula. We have applied this recursion equation to the problem of enumerating single-source acyclic orientations and obtained at an algorithm for this problem with a better delay than the approach recently proposed in the literature. Moreover, we further simplified the resulting algorithm and have arrived at a particular lean implementation for addressing this enumeration problem.

Our discussion of the algorithms suggests that further research on modifications of the algorithms may be an interesting and useful path to pursue. Given a specific structure of the graph whose single-source acyclic orientations are to be enumerated, it may be possible to sort the edges beforehand such that a check for connectedness after deleting an edge that has previously been contracted may turn out to be very simple. In the case of bipolar orientations, specific graph structures may also lead to a situation similar to the single-source acyclic case where we can know feasible contractions beforehand. If this cannot be achieved for a particular graph structure, a less ambitious aim for further research would be to study the conditions under which the edges can be selected as to minimize σ, the total number of edges to be carried over when trying whether a contraction is feasible.

Finally, let us point out that the paper by Fraysseix et al. [12] implies another option for enumerating bipolar (and, by implication, single-source acyclic) orientations in the special case of 3-connected graphs. Fraysseix et al. have shown that in this case, for each bipolar orientation there always exists a single edge whose orientation can be reversed such that we arrive at another bipolar orientation, and that all bipolar orientations can be reached from one bipolar orientation by sucessively carrying out such a single reversal of orientation that preserves the bipolar property. Combining this insight with the backward search technique proposed by Avis and Fukuda [3] would lead to another new approach of enumerating bipolar and single-source acyclic orientations for 3-connected graphs.

References

[1] A. Marino A. Conte, R. Grossi and R. Rizzi. Efficient enumeration of graph orientations with sources. *Discrete Applied Mathematics*, 246:22–37, 2018.

[2] S. Even A. Lempel and I. Cederbaum. An algorithm for planarity testing on graphs. *Theory of Graphs - International Symposium*, pages 215–232, 1967.

[3] David Avid and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65:21–46, 1996.

[4] Norman Biggs. The tutte polynomial as a growth function. *Journal of Algebraic Combinatorics*, 10:115–133, 1999.

[5] Deeparnab Chakrabarty Brian Benson and Prasad Tetali. G-parking functions, acyclic orientations and spanning trees. *Discrete Mathematics*, 310:1340–1353, 2010.

[6] Giuseppe F. Italiano David Eppstein, Zvi Galil and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *Journal of the ACM*, 44(5):669–696, 1997.

[7] Reinhard Diestel. *Graph Theory*. Springer, Berlin, 2017.

[8] J. Ebert. st-ordering the vertices of biconnected graphs. *Computing*, 30:19–33, 1983.

[9] Shimon Even and Robert Endre Tarjan. Computing an st-numbering. *Theoretical Computer Science*, 2:339–344, 1976.

[10] Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM Journal of Computing*, 14(4):781–798, 1985.

[11] Curtis Greene and Thomas Zaslavsky. On the interpretation of whitney numbers through arrangements of hyperplanes, zonotypes, non-radon partitions, and orientations of graphs. *Transactions of the American Mathematical Society*, 280(1):97–108, 1983.

[12] P. Rosenstiehl H. de Fraysseix, P. Ossona de Mendez. Bipolar orientations revisited. *Discrete Applied Mathematics*, 56:157–179, 1995.

[13] Charalampos Papamanthou and Ioannis G. Tollis. Applications of parametrized st-orientations in graph drawing algorithms. In P. Healy and N.S. Nikolov, editors, *Graph Drawing. GD 2005. Lecture Notes in Computer Science 3843*, pages 355–367. Springer, Berlin, 2005.

[14] C. R. Platt. Planar lattices and planar graphs. *Journal of Combinatorial Theory (B)*, 21:30–39, 1976.

[15] Pierre Rosenstiehl and Robert Endre Tarjan. Rectilinear planar layouts and bipolar orientations of planar graphs. *Discrete and Computational Geometry*, 1:343–353, 1986.

[16] J. M. Schmidt. A simple test on 2-vertex- and 2-edge-connectivity. *Information Processing Letters*, 113(7):241–244, 2013.

[17] Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–390, 1983.

[18] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2):146–160, 1972.

[19] Robert Endre Tarjan. Two streamlined depth-first search algorithms. *Fundamenta Informaticae*, IX:85–94, 1986.

[20] W.T. Tutte. *Graph Theory*, volume 21. Addison-Wesley, Reading, MA, 1984.