Efficient enumeration of graph orientations with sources

Alessio Conte, Roberto Grossi, Andrea Marino, Romeo Rizzi

Efficient enumeration of graph orientations with sources

Alessio Conte [a], Roberto Grossi [a], Andrea Marino [a,*], Romeo Rizzi [b]

[a] *Università di Pisa and Erable, Inria, Italy*
[b] *Università di Verona, Italy*

Keywords:
Enumeration
Acyclic orientations
Cyclic orientations
Delay

A B S T R A C T

An orientation of an undirected graph is obtained by assigning a direction to each of its edges. It is called cyclic when a directed cycle appears, and acyclic otherwise. We study efficient algorithms for enumerating the orientations of an undirected graph. To get the full picture, we consider both the cases of acyclic and cyclic orientations, under some rules specifying which nodes are the sources (i.e. their incident edges are all directed outwards). Our enumeration algorithms use linear space and provide new bounds for the delay, which is the maximum elapsed time between the output of any two consecutively listed solutions. We obtain a delay of $O(m)$ for acyclic orientations and $\tilde{O}(m)$ for cyclic ones. When just a single source is specified, these delays become $O(m \cdot n)$ and $O(m \cdot h + h^3)$, respectively, where h is the girth of the graph without the given source. When multiple sources are specified, the delays are the same as in the single source case.

1. Introduction

An orientation of an undirected graph is any of the directed graphs that may be obtained by assigning directions to its edges. As such, the terminology of directed graphs applies to orientations: they are called cyclic when containing a directed cycle, and acyclic otherwise. Their sources are the nodes whose incident edges are all directed outwards. Acyclic orientations have been studied in depth. Motivated by the fact that each acyclic orientation corresponds to a partial order for the underlying graph, Iriarte [14] investigated which orientations maximize the number of linear extensions of the corresponding poset. Alon and Tarsi [1] looked for special orientations to give bounds on the size of the maximum independent set or the chromatic number. Gallai, Roy, and Vitaver independently stated that every orientation of a graph with chromatic number k contains a simple directed path with k nodes [11,21,29]. Johnson [16] links acyclic orientations with exactly one source to problems of network reliability.

There are further problems that can be addressed by looking at acyclic orientations. For instance, Benson et al. [5] showed that there exists a bijection between the set of the so-called superstable configurations of a graph and the set of its acyclic orientations with a unique source. Counting the number of acyclic orientations is a problem dating back to the 70 s or earlier [26], and Linial [18] proved that this problem is #P-complete. Stanley [25] showed how to count them using the chromatic polynomial (a special case of Tutte's polynomial). A related problem is the acyclic orientation game. Alon and Tuza [2] inquired about the amount of oriented edges needed to define a unique orientation, and found this number to be almost surely $\Theta(n \log n)$ in Erdős–Rényi random n-node graphs. Pikhurko [20] showed that the number of these edges in the worst case is no greater than $(\frac{1}{4} + o(1))n^2$ in general.

* Corresponding author.
E-mail addresses: conte@di.unipi.it (A. Conte), grossi@di.unipi.it (R. Grossi), marino@di.unipi.it (A. Marino), rizzi@di.univr.it (R. Rizzi)

Cyclic orientations have received less attention. Counting them is clearly #P-complete.[1] Fisher et al. [13] studied the number of dependent edges, i.e. edges generating a cycle if reversed in an orientation. This number of edges implicitly gives a hint about the number of cyclic orientations in a graph. The related problem of enumerating *strong orientations*, a special type of cyclic orientations, was also considered in [9].

We think that cyclic orientations have interesting properties to study, even though they seem more artificial and have much less applications than acyclic ones. However they can hopefully contribute to give a full picture, as acyclic and cyclic orientations correspond to an exact partition of all the 2^m possible orientations of a graph. As listing all orientations of a graph is straightforward, one might think that solving one problem yields the result for the other. This however is not at all satisfactory as we will see. Furthermore, techniques and invariants for one problem do not apply to the other: acyclic orientation can be obtained incrementally by keeping acyclicity as an invariant. This can be seen as a somewhat "global" property with no proper translation for the cyclic case: cyclic orientation are based on the "local" property of the existence of a directed cycle somewhere in the graph, and this cycle may appear at any time during an incremental approach. Ensuring the satisfiability of this "local" cycle-existence property, while seemingly easy, appears to yield slower algorithms than for the acyclic case. This hints at the possibility of the problem being harder, or simply that stronger techniques are required.

Problems studied. Our paper considers both acyclic and cyclic orientations, investigating how to enumerate them. We will use the terms enumerating and listing interchangeably. We are given an undirected connected graph $G(V, E)$ without self loops, where $n = |V|$ and $m = |E|$, and we will study the following cases.

Single Source Orientations (SSO): Given a node $s \in V$, enumerate all the orientations of G, such that s is the only source.
Single Source Acyclic Orientations (SSAO): Given a node $s \in V$, enumerate all the acyclic orientations of G, such that s is the only source.
Acyclic Orientations (AO): Enumerate all the acyclic orientations of G.
Single Source Cyclic Orientations (SSCO): Given a node $s \in V$, enumerate all the cyclic orientations of G, such that s is the only source.
Cyclic Orientations (CO): Enumerate all the cyclic orientations of G.

As we will see the above variations of orientations provide an interesting playground for exploring enumerations algorithms on graphs. We propose new efficient algorithms and analyze their cost in terms of *delay*, which is a well-known measure of performance for enumeration algorithms corresponding to the worst-case time between any two consecutively enumerated solutions (e.g. [17]). We will focus on algorithms with guaranteed delay and space. Furthermore, we show in Section 8 how the proposed algorithms can be used to solve a wider range of orientation listing problems, in particular with respect to multiple sources.

Previous work. We are not aware of any listing algorithm for SSO. The result of [27] gives necessary and sufficient conditions for the existence of a single source orientation and a search algorithm to find few of them. However, this algorithm cannot be easily revised for enumeration purposes.

Problem AO has been investigated by Squire [24]. His algorithm has a good amortized cost of $O(n)$ time per solution, but the delay can be $O(n^3)$ time. The algorithm by Barbosa and Szwarcfiter [4] solves AO with an amortized time complexity of $O(n + m)$ per solution, and the delay is $O(n \cdot m)$. An alternative way of solving AO is by applying any algorithm for maximal feedback arc set enumeration (after replacing each edge with a double arc). State of the art approaches for the latter problem incur in a delay of $\Omega(n^3)$ as shown by Schwikowski and Speckenmeyer [22].

It should be noted that AO has a correspondence with cell enumeration in an arrangement of hyperplanes, which has been solved in the paradigm of reverse search [3]. Even though such algorithm can be designed to be memory efficient, e.g. by employing the techniques in [10], its delay is $\Omega(n \cdot m)$.

SSAO seems to be harder to solve, as all the techniques above for AO (including the one in [24]) do not extend smoothly. Johnson [16] presented a backtracking algorithm for SSAO, aimed at solving problems on network reliability. However its complexity is hard to estimate, as it is based on a backtracking approach with dead ends. For this reason Squire [24] writes that he has been unable to efficiently implement Johnson's approach. As far as we know, there are no provably good bounds in the literature for problems SSAO and their variants, other than the special case in which G is a planar biconnected graph, and a single target t adjacent to s is allowed [23].

As for SSCO and CO, the best known method is to enumerate them by difference: go through all possible 2^m orientations and eliminate, say, the α acyclic ones. This method does not guarantee polynomial delay as the number $\beta = 2^m - \alpha$ of cyclic orientations can be much larger or much smaller than α: for example, a tree with m edges has $\alpha = 2^m$ and $\beta = 0$. On the other hand, a clique with n nodes and $m = \frac{n \cdot (n-1)}{2}$ edges has $\alpha = n!$, i.e. the possible transitive tournaments [19], and $\beta = 2^m - n!$. As 2^m grows faster than $n!$, the ratio $\alpha/\beta = n!/(2^m - n!)$ tends to 0 for increasing n.

Results. For SSO and SSAO we design the first enumeration algorithms with guaranteed delay of $O(m)$ and $O(m \cdot n)$, respectively, using $O(m)$ space. For AO, we show how to obtain $O(m)$ delay: even if the latter result does not improve the amortized cost of $O(n)$ in [4,24], it improves their delay and that of [22]. For CO and SSCO, we provide the first enumeration algorithms with guaranteed delay, respectively, of $\tilde{O}(m)$ and $O(m \cdot h + h^3)$, where $h < n$ is the girth (length of the shortest

1 For a graph with m edges, there are 2^m orientations, which are either cyclic or acyclic: we have seen that counting the latter ones is #P-complete [18].

Table 1
Summary of the delay of our listing algorithms and that of state of the art algorithms, where m and n are respectively the number of edges and nodes of the graph, and h is the girth of the graph without the given source s.

Problem	Previous work	This work
SSO	Unknown	$O(m)$
SSAO	Unbounded [17]	$O(m \cdot n)$
AO	$O(n^3)$ [4]	$O(m)$
SSCO	Unknown	$O(m \cdot h + h^3)$
CO	Unknown	$\tilde{O}(m)$

cycle) of the graph without the given source. Results are summarized in Table 1. These problems have versions with multiple sources, instead of single, that can be easily solved by our algorithms (see Sections 8) with the same delay. A preliminary version of some of the results appeared in Conte et al. (2015, 2016) [7,8]. Our paper is divided into two parts. Part I describes the algorithms for SSO, SSAO and AO in Sections 3–5, while Part II describes those for SSCO and CO in Section 6–7. Finally, in Section 8 we show how to solve some variations of these enumeration problems.

2. Preliminaries

Given an undirected graph $G(V, E)$ with n nodes and m edges, an orientation of G is the directed graph $\overrightarrow{G}(V, \overrightarrow{E})$ where for any pair $\{u, v\} \in E$ either $(u, v) \in \overrightarrow{E}$ or $(v, u) \in \overrightarrow{E}$. We call $\overrightarrow{E}$ an orientation of E. We say that the orientation $\overrightarrow{G}$ is acyclic when it does not contain cycles, or cyclic otherwise. For the sake of clarity, in the following we will call *edges* the unordered pairs $\{x, y\}$ (undirected graph), while we will call *arcs* the two possible orientations (x, y) and (y, x) (directed graphs). We assume without loss of generality that G is connected and does not contain self-loops.

Given an undirected graph $G(V, E)$, let $v_1, \ldots, v_n \in V$ be an arbitrarily fixed ordering of the nodes of G. We define $V_{\leq i}$ as the set $\{v_1, \ldots, v_i\}$, $N(v) = \{x : \{v, x\} \in E\}$ as the set of neighbors of the node v_i, and $N_{\leq i}(v)$ as the restricted set $N(v) \cap V_{\leq i}$. For brevity, $N_<(v_j)$ means $N_{\leq j-1}(v_j)$. Clearly we have $\sum_{j=1}^{n} |N_<(v_j)| = m$.

PART I: ALGORITHMS FOR ACYCLIC ORIENTATIONS

The algorithms for SSO, SSAO and AO follow a common scheme. Starting from an empty directed graph $\overrightarrow{G}_0$, they add v_i to the previous directed graph $\overrightarrow{G}_{i-1}$ for increasing values of $i = 1, 2, \ldots, n$. For this, we define a *direction assignment* $\overrightarrow{Z}_i$ as an orientation of the set of edges $\{\{v_i, x\} : x \in N_<(v_i)\}$. We denote two special direction assignments by

$$X_i = \{(x, v_i) : x \in N_<(v_i)\}$$

$$Y_i = \{(v_i, x) : x \in N_<(v_i)\}.$$

The algorithms for SSO, SSAO and AO explore only the "valid" direction assignments to obtain the resulting directed graphs $\overrightarrow{G}_i$: every time node v_n is reached, the corresponding $\overrightarrow{G}_n$ is an orientation of G and thus is output as a new solution $\overrightarrow{G}$. The algorithms employ different definitions of "valid", as discussed in Sections 3–5.

3. Single source orientations (SSO)

We illustrate the basic scheme in some detail. We recall that in SSO the orientations $\overrightarrow{G}$ must have s as their only source. To avoid dead ends, we exploit a suitable ordering of the nodes.

Definition 1 (*full Node*)**.** For $1 \leq j \leq i$ a node v_j is *full* in $\overrightarrow{G}_i$ if $N_{\leq i}(v_j) = N(v_j)$. We will also simply call a node *full* if $\overrightarrow{G}_i$ is clear from the context.

Definition 2 (*Valid Direction Assignment*)**.** Given $\overrightarrow{G}_{i-1}(V_{\leq i-1}, \overrightarrow{E})$, the direction assignment $\overrightarrow{Z}_i$ is *valid* if every full node $v_j \neq s$ in $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{Z}_i)$ is *not* a source, for $1 \leq j \leq i$.

The rationale is the following. When dealing with $\overrightarrow{G}_i$, there are no unassigned incident edges for the full nodes: if any of them is a source, it will remain a source in the final orientation $\overrightarrow{G}$. Hence, only s can be a full node that is a source. To guarantee this, we order the nodes suitably.

Definition 3. An ordering of the nodes $v_1, \ldots, v_n$ is *good* if

- $v_n = s$, and
- $N_<(v_i) \neq N(v_i)$, for $1 \leq i < n$.

Algorithm 1: `single-source-orientations`

Input: $G(V, E)$, partial orientation $\overrightarrow{G}_{i-1}(V_{\leq i-1}, \overrightarrow{E})$, integer i
Output: Orientations of G containing $\overrightarrow{G}_{i-1}(V_{\leq i}, \overrightarrow{E})$ with source s

if $i > n$ **then** output $\overrightarrow{G}$; **return**
for any valid direction assignment $\overrightarrow{Z}_i$ for v_i starting from $\overrightarrow{Z}_i = Y_i$ **do**
  `single-source-orientations`$(G, \overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{Z}_i), i+1)$

Algorithm 2: Returning valid direction assignments for sso

Input: Graph $G(V, E)$, partial orientation $\overrightarrow{G}_{i-1}(V_{\leq i-1}, \overrightarrow{E})$, node v_i
Output: Valid direction assignments $\overrightarrow{Z}_i$

$F_i \leftarrow$ set of full nodes in $\overrightarrow{G}_i$ that are sources and not full in $\overrightarrow{G}_{i-1}$
$\overrightarrow{Z}_i \leftarrow \{(v_i, y) : y \in F_i\}$
Let $x_1, \ldots, x_k$ be the nodes in $N_{\leq}(v_i) \setminus F_i$
Execute `Generate`$(G, \overrightarrow{G}, v_i, \overrightarrow{Z}_i, 1)$.

Procedure `Generate`$(G(V, E), \overrightarrow{G}_{i-1}(V_{\leq i-1}, \overrightarrow{E}), v_i, \overrightarrow{W}, j)$
  if $j > k$ **then** add $\overrightarrow{W}$ to the output list; **return**
  `Generate`$(G, \overrightarrow{G}_{i-1}, v_i, \overrightarrow{W} \cup \{(v_i, x_j)\}, j+1)$
  `Generate`$(G, \overrightarrow{G}_{i-1}, v_i, \overrightarrow{W} \cup \{(x_j, v_i)\}, j+1)$

The first condition in Definition 3 says that s should be the last node as it is the only source. The second condition says that there is at least one unassigned incident edge for each added v_i. We avoid dead ends when adding v_i to $\overrightarrow{G}_{i-1}$, as at least one solution extending $\overrightarrow{G}_{i-1}$ exists such that v_i is not a source. To see why, let us call $\overrightarrow{G}_i$ a partial orientation (of G) and give the following property.

Property 1. *For any partial orientation $\overrightarrow{G}_i$, there is always an orientation $\overrightarrow{G}$ for G that has unique source s and extends $\overrightarrow{G}_i$.*

Proof. Consider the direction assignments Y_j ($j > i$) and the resulting acyclic $\overrightarrow{G}$. These direction assignments are valid as they cannot create new sources due to the good ordering, thus the only final source in $\overrightarrow{G}$ is s. □

A good ordering for G and s can be found in linear time by performing a DFS from s and considering its nodes in postorder. Observe that this is a good order according to our definition: s is the last node and, for each node, its parent in the DFS tree appears after it in the order.

Algorithm 1 details how to proceed according to the good ordering of the nodes. The initial call is `single-source-orientations`$(G, \overrightarrow{G}_0, 1)$. The algorithm recursively explores all the possible ways of expanding the current partial solution $\overrightarrow{G}_{i-1}$ by iterating over all the valid direction assignments, starting from Y_i, which is surely valid by Property 1. The assignments are generated by a recursive computation. We have the primary recursion tree to generate all the wanted orientations (Algorithm 1), where each node has associated a secondary recursion tree to generate locally all the valid direction assignments as explained next.

Iterating over valid direction assignments.

Algorithm 2 generates all and only the valid direction assignments $\overrightarrow{W}$, for a given $\overrightarrow{G}_{i-1}$ and a node v_i, among the possible $2^{|N_<(v_i)|}$ ones. Let F_i be the set of nodes that are sources and not full in $\overrightarrow{G}_{i-1}$ and that become full in $\overrightarrow{G}_i$, where $F_i \subseteq N_<(v_i)$. All the valid direction assignments should guarantee that nodes in F_i are not sources in $\overrightarrow{G}_i$: this can be easily done by adding the arcs of $\overrightarrow{Z}_i$ to $\overrightarrow{W}$ (and this is mandatory as s is the only source). After that, we have to decide the orientation of the remaining edges, involving v_i and the nodes in $N_<(v_i) \setminus F_i$. This part relies on procedure `Generate`. In particular, we have to assign a direction to the edges $\{v_i, x_j\}$ for each node x_j in $N_<(v_i) \setminus F_i$: both the directions (v_i, x_j) and (x_j, v_i) are explored.

Since both F_i and $N_<(v_i) \setminus F_i$ are subsets of $N(v_i)$, Lemma 1 holds.

Lemma 1. *Algorithm 2 finds valid direction assignments with delay $O(|N(v_i)|)$.*

Lemma 2. *Referring to Algorithms 1 and 2, the following holds.* (1) *All the orientations of G whose unique source is s are output*; (2) *only the orientations of G whose unique source is s are output*; (3) *there are no duplicates.*

Proof. We prove the three statements separately using the good ordering of the nodes. (1) Consider the following process. For decreasing values of j remove v_j from $\overrightarrow{G}$, and let $\overrightarrow{W}_j$ be the set of the edges incident to v_j. Note that $\overrightarrow{W}_j$ is valid, and so

Algorithm 3: Returning valid direction assignments for SSAO

Input: Graph $G(V, E)$, partial acyclic orientation $\overrightarrow{G}_{i-1}(V_{\leq i-1}, \overrightarrow{E})$, node v_i
Output: Valid direction assignments $\overrightarrow{Z}_i$

$F_i \leftarrow$ set of full nodes in $\overrightarrow{G}_i$ that are sources and not full in $\overrightarrow{G}_{i-1}$
$\overrightarrow{Z}_i \leftarrow \{(v_i, y) \,:\, y \in F_i\}$
Let $x_1, \ldots, x_k$ be the nodes in $N_<(v_i) \setminus F_i$
Execute `Generate` $(G, \overrightarrow{G}, v_i, \overrightarrow{Z}_i, 1, \emptyset, \emptyset)$

Procedure `Generate` $(G(V, E), \overrightarrow{G}_{i-1}(V_{\leq i-1}, \overrightarrow{E}), v_i, \overrightarrow{W}, j, R, B)$
  if $j > k$ **then** add $\overrightarrow{W}$ to the output list; **return**
  Update B as the set of nodes leading to v_i in $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{W})$
  if $x_j \notin B$ **then** `Generate` $(G, \overrightarrow{G}_{i-1}, v_i, \overrightarrow{W} \cup \{(v_i, x_j)\}, j+1, R, B)$
  Update R as the set of nodes reachable from v_i in $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{W})$
  if $x_j \notin R$ **then** `Generate` $(G, \overrightarrow{G}_{i-1}, v_i, \overrightarrow{W} \cup \{(x_j, v_i)\}, j+1, R, B)$

$\overrightarrow{W}_1, \ldots, \overrightarrow{W}_n$ lead from $\overrightarrow{G}_0$ to the discovery of $\overrightarrow{G}$. (2) Node $s = v_n$ is the unique source. For any given $\overrightarrow{G}_i$, each full node is not a source and each non-full node v_j with $j \leq i$ has a neighbor $k > i$ such that $(v_k, v_j) \in Z_k$. Thus the only source is the last node $v_n = s$. (3) It follows from the ordering of the nodes and the valid direction assignments. □

Lemma 3. *Algorithm 1 has delay $O(m)$ and uses space $O(m)$.*

Proof. We exploit the properties of the primary recursion tree induced by Algorithm 1: notice that each internal node has at least one child because of Property 1. This means that all the leaves correspond to a solution. The depth of the recursion tree is $O(n)$ as one new node is considered at each step. Thus, the delay of Algorithm 1 is bounded by the sum of the costs along a leaf-to-root and root-to-next-leaf path. This includes the cost of finding the first solution. Both paths consist in n nested calls of Algorithm 1, whose delay is dominated by the delay of Algorithm 2. As each call considers a different node v_i, and the delay of Algorithm 2 is $O(|N(v_i)|)$ by Lemma 1, we have that the total delay is $O(\sum_{v \in V} |N(v)|) = O(m)$. The space requirement is bounded by the information stored in a root-to-leaf path of the primary recursion. The graph takes $O(m)$ space. The working space is dominated by the `Generate` iterators (calls to Algorithm 2), that is, all direction assignments $\overrightarrow{W}_1, \ldots, \overrightarrow{W}_n$ on the root-to-leaf path, and is bounded by $O(m)$. □

4. Single Source Acyclic Orientations (SSAO)

For a given node s in the graph G, we now want to enumerate the acyclic orientations $\overrightarrow{G}$ such that s is the unique source. We apply the scheme of Section 3 with a different definition of valid direction assignment: it does not create cycles and each full node (except s) is not a source.

Definition 4 (*Valid Direction Assignment*)**.** Given $\overrightarrow{G}_{i-1}(V_{\leq i-1}, \overrightarrow{E})$, the direction assignment $\overrightarrow{Z}_i$ is *valid* if

- $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{Z}_i)$ is acyclic, and
- any $v_j \neq s$ that is *full* in $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{Z}_i)$ is *not* a source, for $1 \leq j \leq i$.

Given the good ordering in Definition 3, the following property, which is an adaptation of Property 1, holds.

Property 2. *For any partial acyclic orientation $\overrightarrow{G}_i$, there is always an acyclic orientation $\overrightarrow{G}$ for G that has unique source s and includes $\overrightarrow{G}_i$.*

Proof. Consider the special assignments Y_j for $j > i$. The only final source is s and no cycle is created as these Y_j's induce a total ordering. □

To solve SSAO we reuse Algorithm 1 and introduce Algorithm 3 that is obtained from Algorithm 2 by a different version of `Generate` described next.

4.1. Iterating over valid direction assignments

Consider the generic step when adding node v_i. We have to include the arcs from $\overrightarrow{Z}_i$ to form $\overrightarrow{W}$ and, for the remaining edges, apply `Generate`. The latter is in charge of assigning a direction to the edges $\{v_i, x_j\}$ for each of the nodes $x_1, \ldots, x_k \in N_<(v_i) \setminus F_i$, and check if they do not create cycles and sources different from s. Namely, for $j = 1, 2, \ldots, k$, if the arc $e \in \{(v_i, x_j), (x_j, v_i)\}$ does not create a cycle in $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{W})$, `Generate` proceeds recursively with $\overrightarrow{W} = \overrightarrow{W} \cup \{e\}$.

We need to perform efficiently the reachability tests for x_j to update sets B and R, where B corresponds to nodes that can lead to v_i, while R corresponds to the ones reachable from v_i. A naive approach requires $O(k)$ DFS traversals in total $O(k \cdot m)$ time. We show how to reduce the latter cost to $O(m)$ time by truncating the DFSes whenever they touch a node in B or R. Since the partially built directed graph is acyclic, B and R are disjoint, and a node can belong to either one of them or none of them. Below we provide an analysis based on a simple coloring scheme.

Lemma 4. *Algorithm 3 returns valid direction assignments with delay $O(m)$.*

Proof. The arcs in $\overrightarrow{Z}_i$ can be computed in $O(m)$ time at the beginning. We discuss the rest of the arcs (i.e. $\overrightarrow{W} \setminus \overrightarrow{Z}_i$). We have to decide whether (v_i, x_j) or (x_j, v_i) creates a cycle or not, and we color incrementally the nodes for this purpose: all the nodes R reachable from v_i are *red*; all the nodes B that can lead to v_i are *black*; the remaining nodes are *uncolored*. Since $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{W})$ is acyclic, any node has just one color or is uncolored.

Initially, all the nodes are uncolored, and R and B are empty. We show that the sum of the costs to update R and B to produce a solution through the recursive calls of `Generate` is $O(m)$. Since each leaf in the secondary recursion tree induced by `Generate` corresponds to a distinct solution, we should bound the sum of the costs along the $k+1$ nodes from the root to that leaf. Specifically, the delay is upper bounded by the sum of the costs along two paths: (a) the leaf-to-root path of the current solution and (b) the root-to-next-leaf path for the next solution (actually only the latter for the first solution).

Observe that the cost of (a) is always $O(|N_<(v_i)|)$. As for (b), it is bounded by $O(m)$ as follows. When $j = 1$, the red colors are assigned with a forward traversal and the black colors are assigned with a backward traversal in the graph $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{W})$. When $j > 1$, while adding the arc (v_i, x_j) to $\overrightarrow{W}$ we have only to make the traversed uncolored nodes red: since the forward traversal is rooted at v_i, we continue the traversal avoiding to visit red nodes. (No black node can be reached, otherwise $\overrightarrow{G}_i$ would be cyclic). On the other hand, when adding the arc (x_j, v_i) to $\overrightarrow{W}$ we only have to make the traversed uncolored nodes black: once again, this corresponds to continuing the backward traversal rooted in v_i avoiding to visit black nodes (no red node can be reached). Since this process traverses each arc at most once for any $1 \leq j \leq k$, the sum of the costs of a root to leaf path in the secondary recursion tree induced by the `Generate` procedure is $O(m)$. □

Remark 1. After the last valid direction assignment has been returned, Algorithm 3 recognizes that there are no more valid direction assignments, using extra $O(m)$ time.

As it can be seen in Algorithm 1, we employ `Generate` as an iterator through all the valid direction assignments $\overrightarrow{Z}_i$. Its state consists in the last direction assignment returned and the information carried by R and B.

Lemma 5. *The state of the `Generate` iterator described in Algorithm 3 can be rebuilt in $O(m)$ time from the last direction assignment returned.*

Proof. We need to mark for each node which edge in $\overrightarrow{Z}_i$ caused it to be added to R or B. This can be done with a DFS rooted in v_i in which edges leaving v_i are visited in the same order as they were added to $\overrightarrow{Z}_i$; each time a new node is visited, the last edge of $\overrightarrow{Z}_i$ traversed was the one that caused it to be added to the set. For B the visit is done by following the edges backwards. The state of the iterator is thus restored in the time required by a DFS, i.e. $O(m)$. □

Lemma 6. *Referring to Algorithm 1 and Algorithm 3, the following holds.*

(1) *All the acyclic orientations of G whose unique source is s are output.*
(2) *Only the acyclic orientations of G whose unique source is s are output.*
(3) *There are no duplicates.*

Proof. Similar to the proof of Lemma 2. □

Theorem 1. *Problem SSAO can be solved with delay $O(n \cdot m)$ and space $O(m)$.*

Proof. We solve the problem with Algorithm 1, but using Algorithm 3 instead of Algorithm 2 to generate the valid direction assignments. The proof is similar to that of Lemma 3. Again, the depth of the tree is $O(n)$, and each node leads to at least a solution (Property 2), meaning that each leaf is a solution. It follows that the delay is similarly bounded by the sum of the costs along a leaf-to-root and root-to-next-leaf path. The former is bounded by $O(n \cdot m)$: the height of the tree is $O(n)$ and each return we spend $O(m)$ to recognize that no more valid direction assignments are possible, as highlighted by Remark 1. The latter is still bounded by $O(n \cdot m)$, applying n times Lemma 4, for a total delay of $O(n \cdot m)$. As for Algorithm 1 the space requirement is bounded by the information stored in a root-to-leaf path of the primary recursion. The input data and the sum of all direction assignments take $O(m)$ space. The state of `Generate` iterator takes $O(n)$ space for each node, but it does not need to be stored as it can be rebuilt from the last direction assignment generated when backtracking, as stated by Lemma 5. As rebuilding the state is done once for node and takes $O(m)$ time, this procedure takes $O(n \cdot m)$ which does not affect the total delay. Hence the algorithm has delay $O(n \cdot m)$ and space requirement $O(m)$. □

Algorithm 4: Returning valid direction assignments for AO

Input: Graph $G(V, E)$, partial acyclic orientation $\overrightarrow{G}_{i-1}(V_{\leq i-1}, \overrightarrow{E})$, node v_i
Output: Valid direction assignments $\overrightarrow{Z}_i$

Let $x_1, \ldots, x_k$ be the nodes of $N_<(v_i)$
Execute `Generate` $(G, \overrightarrow{G}, v_i, \emptyset, 1, \emptyset, \emptyset)$

Procedure `Generate` $(G(V, E), \overrightarrow{G}_{i-1}(V_{\leq i-1}, \overrightarrow{E}), v_i, \overrightarrow{W}, j, R, B)$
  if $j \geq k$ **then** add $\overrightarrow{W}$ to the output list; **return**
  if $\overrightarrow{W} \cap Y_i \neq \emptyset$ **then**
    update R as the nodes reachable from v_i in $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{W})$
  if $x_j \notin R$ **then** `Generate` $(G, \overrightarrow{G}_{i-1}, v_i, \overrightarrow{W} \cup \{(x_j, v_i)\}, j+1, R, B)$
  if $\overrightarrow{W} \cap X_i \neq \emptyset$ **then**
    update B as the nodes leading to v_i in $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{W})$
  if $x_j \notin B$ **then** `Generate` $(G, \overrightarrow{G}_{i-1}, v_i, \overrightarrow{W} \cup \{(v_i, x_j)\}, j+1, R, B)$

5. Acyclic Orientations (AO)

Differently from Sections 3–4, we have no restrictions about the possible sources when adding v_i. Hence we define the concept of valid direction assignment as follows.

Definition 5 (*Valid Direction Assignment*)**.** Given $\overrightarrow{G}_{i-1}(V_{\leq i-1}, \overrightarrow{E})$, a direction assignment $\overrightarrow{Z}_i$ is *valid* if $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{Z}_i)$ is acyclic.

Another difference from the previous section is that we do not need the good order (Definition 3). Namely, for any order of the nodes we can prove that the following property holds.

Property 3. *For any ordering $v_1, v_2 \ldots v_n$ of the nodes, given the directed acyclic graph $\overrightarrow{G_{i-1}}$ and the node v_i, there are always at least two valid direction assignments for v_i, i.e. $X_i = \{(x, v_i) : x \in N_<(v_i)\}$ and $Y_i = \{(v_i, x) : x \in N_<(v_i)\}$.*

Proof. Trivial, as v_i becomes either a source or a sink (i.e. its incident edges are all directed inwards) in $\overrightarrow{G_i}$, thus cycles are not created. By induction G can be oriented completely. □

Algorithm 4 provides a way of iterating over valid direction assignments. It explores all of these similarly to Algorithm 3. Notice that the first valid direction assignment produced is X_i and the last valid direction assignment is Y_i. Moreover observe that the update of R and B is respectively not required when $\overrightarrow{W} \cap Y_i = \emptyset$ and $\overrightarrow{W} \cap X_i = \emptyset$, since these conditions mean respectively that the outdegree and the indegree of v_i is zero in $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{W})$. Summing up, the overall scheme remains the same as that of Algorithm 1. By Property 3, $\overrightarrow{Z}_i = X_i$ and $\overrightarrow{Z}_i = Y_i$ are always taken. The corresponding two recursive calls are done respectively at the beginning and at the end of the procedure. All the other valid direction assignments (if any) are explored in the other calls of the `for` cycle.

Lemma 7. *Referring to Algorithm 1 and Algorithm 4, the following holds.*

(1) *All the acyclic orientations of G are output.*
(2) *Just the acyclic orientations of G are output.*
(3) *There are no duplicates.*

Proof. Similar to the proof of Lemma 2. □

Lemma 8. *Algorithm 4 returns the first valid direction assignment in time $O(|N_<(v_i)|)$, and the remaining ones with delay $O(m)$.*

Proof. Similar to the proof of Lemma 4, except that X_i is returned in time $O(|N_<(v_i)|)$ since the update of R is not needed and never performed for X_i in Algorithm 4. □

By using Lemma 8, we obtain the following.

Theorem 2. *Problem AO can be solved with delay $O(m)$ and space $O(m)$.*

Proof. We employ Algorithm 1 along with Algorithm 4 as iterator for valid direction assignments. The proof is similar to those for SSO and SSAO (see Theorem 1): while we bounded the delay with a leaf-to-root and root-to-next-leaf path, here we consider the *lowest common ancestor* node z between two consecutive leaves a and b (note that z might still be

Algorithm 5: Returning all the cyclic orientations of G

Input: An undirected connected graph $G(V, E)$
Output: All the cyclic orientations $\overrightarrow{G}(V, \overrightarrow{E})$

Algorithm setup (Section 6.1):
$G(V, E) \leftarrow$ apply bridge and degree-0 node removal to G
$M(V_M, E_M) \leftarrow$ chain compression of G
$C(V_C, E_C) \leftarrow$ any log-hole of M
$M'(V_M, E'_M) \leftarrow$ delete C's edges from M (i.e. $E'_M = E_M \setminus E_C$)

Enumerate cyclic orientations (Section 6.2):
for each extended orientation $\overrightarrow{M}'$ of multigraph M' **do**
  for each legal orientation $\overrightarrow{C}$ of log-hole C (see Algorithm 6) **do**
    $\overrightarrow{M}''(V_M, \overrightarrow{E}'') \leftarrow$ combine $\overrightarrow{M}'$ and $\overrightarrow{C}$, where $\overrightarrow{E}'' = \overrightarrow{E}'_M \cup \overrightarrow{E}_C$
    Output each of the cyclic orientations $\overrightarrow{G}$ of G corresponding to $\overrightarrow{M}''$

the root). As every node leads to a leaf/solution, we can assume that the path from a to z is made of *last children* (Y_i), as otherwise there would be another leaf between a and b. For the same reason, the path from z to b is made of *first children* (X_i). As the backtracking cost after a last child is $O(1)$, the cost of selecting the next valid direction assignment in z is $O(m)$, and the cost of recurring in the first child is $O(|N(v_i)|)$ (see Lemma 8), we have that the delay of the algorithm is $O(n + m + \sum_{i \in V} |N(v_i)) = O(m)$. □

PART II: ALGORITHMS FOR CYCLIC ORIENTATIONS

Problems co and ssco address cyclic orientations. The algorithm for co (Section 6) preprocesses the graph in a way that ensures the existence of a cycle C of logarithmic length in the size of the graph. It then considers all the orientations of the residual graph after C's removal, leading to an extended version of co that can assign to each edge a direction or a *broken* state. The algorithm finally reintegrates C in the solutions for the extended version, exploiting all suitable orientations of C. For the sake of completeness, we also show in Section 7 how to solve ssco. One crucial difference with the algorithm for co is the fact that we cannot perform the preprocessing phase so that we have no guarantee about the logarithmic length of C, which negatively affects the running time.

6. Cyclic Orientations (co)

The intuition behind our algorithm is as follows. Suppose that $G(V, E)$ is cyclic, otherwise there are no cyclic orientations. Consider a cycle $C(V_C, E_C)$ in G: we can orient its edges in two ways so that the resulting $\overrightarrow{C}$ is a directed cycle.[2] At this point, *any* orientation of the remaining edges $E_G \setminus E_C$, will give a cyclic orientation of G. Thus we generate all possible orientations of the edges in $E_G \setminus E_C$, and then assign some suitable orientations to the edges in E_C. This guarantees that we have at least two solutions for each orientation of $E_G \setminus E_C$, namely, setting the orientation of E_C so that $\overrightarrow{C}$ is one of the two possible directed cycles. Yet this is not enough as we may have a cyclic orientation even if $\overrightarrow{C}$ is acyclic.

Therefore we have to handle some cases in general. One easy case is that the orientation of $E_G \setminus E_C$ already induces a directed cycle: any orientation of E_C will give a cyclic orientation of G. Another easy case, as seen above, is when E_C is oriented such that $\overrightarrow{C}$ is a directed cycle: any orientation $\overrightarrow{G}$ including $\overrightarrow{C}$ will be cyclic. The remaining case is not trivial: when the orientations of $E_G \setminus E_C$ and E_C are individually acyclic, putting them together might or not induce a cyclic orientation of G. To efficiently deal with the latter case, we need to "massage" G and transform it into a multigraph as summarized in Algorithm 5 and discussed below.

6.1. Algorithm setup

We reduce the problem of enumerating cyclic orientations to an extended version that allows us to neglect bridges and chains.

Bridge and degree-0 node removal. Given an undirected graph $G(V, E)$, a *bridge* is an edge whose removal increases the number of connected components. As no cycle in G can contain a bridge, we can remove all bridges from G. After a cyclic orientation of the remaining edges is produced, all bridges can be reintegrated in the graph with all possible direction assignments.

For the sake of simplicity, we also remove isolated nodes (i.e. nodes of degree zero), so that all remaining nodes have degree 2 or greater. The bridges of a graph can be found in linear time [28]. Finding and removing bridges and removing isolated nodes can be done in $O(m)$ time and space.

[2] We will actually use a chordless cycle of logarithmic size (called log-hole).

Chain compression. Consider a maximal path $v_1, \ldots, v_k$ where v_i has degree 2 (with $2 \leq i \leq k-1$). As the internal nodes of the path have degree 2, this path can only be part of a directed cycle if it is a directed path from v_1 to v_k, or from v_k to v_1. We exploit this property by replacing each of these paths with a single edge, called *chain edge*. A chain edge can be given three direction assignments: (v_1, v_k) and (v_k, v_1), which correspond to the directed paths mentioned above, and *broken*, that is any other direction assignment (which does not induce a directed path). Identifying and compressing all chains can be accomplished in $O(m)$ time by traversing the graph G in a DFS fashion from a node of degree ≥ 3.[3] The output is an undirected multigraph $M(V_M, E_M)$, since it might contain parallel edges or loops. Nodes in $V_M \subseteq V$ are the nodes of V whose degree is ≥ 3, and edges in E_M are the chain edges plus all the edges in E which are not part of a chain. We call the latter ones simple edges to distinguish them from the chain edges. In the rest of the paper, M will be considered a multigraph where $|V_M| \geq 4$ and each of the edges has a label in {simple, chain}. For this, we define the concept of *extended orientation* as follows.

Definition 6 (*Extended Orientation*)**.** For a multigraph $M(V_M, E_M)$ having self loops and edge labels in {simple, chain}, an *extended orientation* $\overrightarrow{M}(V_M, \overrightarrow{E_M})$ is a directed multigraph whose arc set $\overrightarrow{E_M}$ assigns a direction or *broken* to each edge in E_M: in particular, for any simple edge $\{u, v\} \in E_M$, exactly one direction between (u, v) and (v, u) is assigned; for any chain edge $\{u, v\} \in E_M$, either the edge is broken, or exactly one direction between (u, v) and (v, u) is assigned. A directed cycle in $\overrightarrow{M}$ cannot contain a broken edge.

We can explore extended orientations to list cyclic orientations with the following lemma.

Lemma 9. *If we have an algorithm that lists all the extended cyclic orientations of $M(V_M, E_M)$ with delay $f(|E_M|)$, for some $f : \mathbb{N} \to \mathbb{N}$, then we can produce an algorithm that lists all the cyclic orientations of the graph $G(V, E)$ with delay $O(f(|E_M|)+|E|)$.*

Proof. For each extended cyclic orientations $\overrightarrow{M}$ we return a set S of cyclic orientations of G: any simple edge e of $\overrightarrow{M}$ maintains the same direction specified by $\overrightarrow{M}$ in all the solutions in S; for each chain c of $\overrightarrow{M}$, we consider the edges corresponding to c in G, say $e_1, e_2, \ldots, e_h$: if c has a direction in $\overrightarrow{M}$, the same direction of c is assigned to all the edges e_j in all the solutions in S; if c is *broken*, we have to consider all the possible $2^h - 2$ ways of making the path $e_1, e_2, \ldots, e_h$ broken (these are all the possible ways of directing the edges except the only two directing a path). All the solutions in S differ for the way they replace the chain edges.

Getting extended cyclic orientations in $f(|E_M|)$ delay, iterating over all the chain edges c, and iterating over all the corresponding edges of c assigning the specified directions as explained above, we return cyclic orientations of the graph $G(V, E)$ with delay $O(f(|E_M|) + |E|)$. □

Lemma 9 allows us to concentrate on extended cyclic orientations of the labeled multigraph M rather than on cyclic orientations of G. Conceptually, we have to assign binary values (the orientation) to simple edges and ternary values (the orientation or broken) to chain edges. If we complicate the problem on one side by introducing these multigraphs with chain edges, we have a relevant benefit on the other side, as shown next.

Logarithmically bounded hole. A logarithmically bounded hole (hereafter, log-hole) is a chordless cycle whose length is either the *girth* of the graph (i.e. the length of its shortest cycle) or this length plus one.[4]

Given the labeled multigraph obtained in Section 6.1, namely $M(V_M, E_M)$, we perform the following two steps.

(1) **Finding a log-hole.** Find a log-hole $C(V_C, E_C)$ in $M(V_M, E_M)$.
(2) **Removing the log-hole.** Remove the edges in E_C from M, obtaining $M'(V_M, E'_M)$, where $E'_M = E_M - E_C$.

We will use the following well-known result.

Lemma 10 (*Logarithmic Girth [6,12]*)**.** *Let $G(V, E)$ be a graph in which every node has degree at least 3. The girth of G is $\leq 2\lceil \log |V| \rceil$.*

As every node in M has degree ≥ 3, this means that the log-hole C of M has length at most $2\lceil \log |V_M| \rceil + 1$, thus motivating our terminology.

The log-hole C can be found by applying the algorithm in [15], which easily extends to multigraphs, in time $O(|V_M|^2)$: indeed, the algorithm finds a cycle that is either of minimum size, or larger by one. If chords are present in C, we can check whether C includes a smaller cycle and redefine C accordingly in time $O(|C|^2) = O(\log^2 |V_M|)$.

6.2. Enumerating cyclic orientations

We recall that our goal is to list all the cyclic orientations of G. By Lemma 9 this is equivalent to listing the extended cyclic orientations of $M(V_M, E_M)$. We now show that the latter task can be done by suitably combining some orientations from the labeled multigraph $M'(V_M, E'_M)$ and the log-hole $C(V_C, E_C)$ using the following steps.

[3] If no such node exists G is a cycle or a path and the listing problem is trivial.

[4] Minimum cycle means any cycle having minimum number of edges (e.g. a self loop). Chain edges count just one, like simple edges.

1. **Finding extended orientations.** Enumerate all extended orientations (not necessarily cyclic) $\overrightarrow{M'}$ of the multigraph M' (Section 6.3).
2. **Putting back the log-hole.** For each listed $\overrightarrow{M'}(V_M, \overrightarrow{E'}_M)$, consider all the extended orientations $\overrightarrow{C}(V_C, \overrightarrow{E}_C)$ of the log-hole C such that $\overrightarrow{E'}_M \cup \overrightarrow{E}_C$ contains a directed cycle, and obtain the extended cyclic orientations for the multigraph M (Section 6.4).

6.3. Finding extended orientations

This is an easy task. For each edge $\{u, v\}$ in E'_M that is labeled as simple, both the directions (u, v) and (v, u) can be assigned; if $\{u, v\}$ is labeled as chain, the directions (u, v) and (v, u), and broken can be assigned. Each combination of these decisions produces an extended orientation of $M'(V_M, E_M)$. If there are s simple edges and c chain edges in M', where $s + c = |E'_M|$, this generates all possible $2^s 3^c$ extended orientations. Each of them can be easily listed in $O(|E'_M|)$ delay (actually less, but this is not the dominant cost).

6.4. Putting back the log-hole

For each listed $\overrightarrow{M'}$ we have to decide how to put back the edges of the cycle C, namely, how to find the orientations of C that create directed cycles.

Definition 7. Given the cycle $C(V_C, E_C)$ and $\overrightarrow{M}'(V_M, \overrightarrow{E}'_M)$, we call *legal orientation* $\overrightarrow{C}(V_C, \overrightarrow{E}_C)$ any extended orientation of C such that the resulting multigraph $\overrightarrow{M}''(V_M, \overrightarrow{E}'')$ is cyclic, where $\overrightarrow{E}'' = \overrightarrow{E}'_M \cup \overrightarrow{E}_C$.

The two following cases are possible.

1. $\overrightarrow{M}'$ is cyclic. In this case each edge in E_C can receive any direction, including broken if the edge is a chain edge: each combination of these assignments will produce a legal orientation that will be output.
2. $\overrightarrow{M}'$ is acyclic. Since C is a cycle, there are at least two legal orientations obtained by orienting C as a directed cycle clockwise and counter-clockwise. Moreover, adding just an oriented subset of edges $D \subseteq C$ to $\overrightarrow{M}'$ may create a cycle in $\overrightarrow{M}'$: in this case, any orientation of the remaining edges of $C \setminus D$ (including broken for chain edges) will clearly produce a legal orientation.

While the first case is immediate, the second case has to efficiently deal with the following problem.

Problem 1. Given $\overrightarrow{M'}$ acyclic and cycle C, enumerate all the legal orientations $\overrightarrow{C}(V_C, \overrightarrow{E}_C)$ of C.

In order to solve Problem 1, we exploit the properties of C. In particular, we compute the reachability matrix R among all the nodes in V_C, that is, for each pair u, v of nodes in V_C, $R(u, v)$ is 1 if u can reach v in $\overrightarrow{M}$, 0 otherwise. We say that R is *cyclic* whether there exists a pair i, j such that $R(i, j) = R(j, i) = 1$. This step can be done by performing a BFS in $\overrightarrow{M'}$ from each node in V_C: by Lemma 10 we have $|V_C| \leq 2\lceil \log |V_M| \rceil + 1$, and so the cost is $O(|E'_M| \cdot \log |V_M|)$ time. Deciding the orientation of the edges and the chain edges in E_C is done with a ternary partition of the search space described below.

Scheme for legal orientations. The steps are shown in Algorithm 6. At the beginning the reachability matrix R is computed as described above, and passed to the recursive routine `LegalOrientations`. At each step, $\overrightarrow{C}'$ is the partial legal orientation to be completed and I is the set of broken edges declared so far. Also, j is the index of the next edge $\{c_j, c_{j+1}\}$ of the cycle C, with $1 \leq j \leq h$ (we assume $c_{h+1} = c_1$ to close the cycle): if $j = h + 1$ then all the edges of C have been considered and we output the solution $\overrightarrow{C}'$ together with the list I of broken edges in $\overrightarrow{C}'$. Each time the procedure is called we guarantee that the reachability matrix R is updated.

Let $\{u, v\}$ be the next edge of C to be considered, where $u = c_j$ and $v = c_{j+1}$: for each possible direction assignment (u, v) or (v, u) of this edge, we have to decide whether we will be able to complete the solution considering this assignment. This is done by trying to add the arc to the current solution. If there is already a cycle, clearly we can complete the solution. Otherwise, we perform a *reachability check* on $\{c_{j+1}, \ldots, c_{h+1}\}$: it is still possible to create a directed cycle if and only if any two of the nodes in $\{c_{j+1}, \ldots, c_{h+1}\}$, say c_f and c_g satisfy $R(c_f, c_g) = 1$ or $R(c_g, c_f) = 1$. This condition guarantees that a cycle will be created in the next calls, since we know there are edges in C between c_f and c_g that can be oriented suitably. Finally, when $\{u, v\}$ is a chain, the broken assignment is also considered: R does not need to be updated as the broken edge does not change the reachability of $\overrightarrow{M'}$.

We discuss now how to perform the reachability and cyclicity checks. Updating R when adding an arc (u, v) corresponds to making v, and all nodes reachable from v, reachable from u and nodes that can reach u. This can be done by simply performing an `or` between the corresponding rows in time $O(\log^2 |V_M|)$, since R is $|C| \times |C|$. The reachability check can be done in $O(\log^2 |V_M|)$ time. The cyclicity (checking whether a cycle has been already created) takes the same of time by looking for a pair of nodes x', y' in $\{c_1, \ldots, c_j\}$ such that $R(x', y') = R(y', x') = 1$. We also have to restore R but the cost is dominated by the rest.

Algorithm 6: Returning all legal orientations of C

Input: $\overrightarrow{M'}(V_M, \overrightarrow{E'}_M)$ acyclic, a cycle $C(V_C, \underline{E_C})$ with $V_C \subseteq V_M$
Output: All the legal orientations $\overrightarrow{C}(V_C, \overrightarrow{E}_C)$

Build the reachability matrix R for the nodes of V_C in $\overrightarrow{M'}$
Let $V_C = \{c_1, \dots c_h\}$, where $c_{h+1} = c_1$ by definition
Execute `LegalOrientations` $(\overrightarrow{C}'(\emptyset, \emptyset), 1, R, \emptyset)$

Procedure `LegalOrientations`$(\overrightarrow{C}'(V'_C, \overrightarrow{E}'_C), j, R, I)$
- **if** $j = h + 1$ **then**
  - output $\overrightarrow{C}'$ and its set I of broken edges
- **else**
  - $u \leftarrow c_j$, $v \leftarrow c_{j+1}$
  - $R_1 \leftarrow R$ updated by adding the arc (u, v)
  - **if** R_1 *is cyclic or has positive reachability test on* $\{c_{j+1}, \dots, c_{h+1}\}$ **then**
    - `LegalOrientations` $(\overrightarrow{C}'(V'_C, \overrightarrow{E}'_C \cup \{(u, v)\}), j + 1, R_1, I)$
  - $R_2 \leftarrow R$ updated adding the arc (v, u)
  - **if** R_2 *is cyclic or has positive reachability test on* $\{c_{j+1}, \dots, c_{h+1}\}$ **then**
    - `LegalOrientations` $(\overrightarrow{C}'(V'_C, \overrightarrow{E}'_C \cup \{(v, u)\}), j + 1, R_2, I)$
  - **if** $\{u, v\}$ *is a chain edge* **then**
    - **if** R *is cyclic or has positive reachability test on* $\{c_{j+1}, \dots, c_{h+1}\}$ **then**
      - `LegalOrientations` $(\overrightarrow{C}', j + 1, R, I \cup \{\{u, v\}\})$

Lemma 11. *Algorithm 6 outputs in time $O(|E'_M| \log |V_M|)$ the first legal orientation of C, and each of the remaining ones with $O(\log^3 |V_M|)$ delay.*

Proof. Before calling `LegalOrientations` we have to compute the reachability matrix from scratch and this costs $O(|E'_M| \log |V_M|)$ time. In the following we bound the delay between two outputs returned by `LegalOrientations`. Firstly, note that each call produces at least one solution. This is true when $j = 1$ since we have two possible legal orientations of C. Before performing any call at depth j, the caller function checks whether this will produce at least one solution. Only calls that will produce at least one solution are then performed. This gives a recursion tree similar to the ones seen for SSAO, where every internal node has at least one child and each leaf corresponds to a solution. Hence the delay between any two consecutive solutions is bounded by the cost of a leaf-to-root path and the cost of a root-to-the-next-leaf path in the recursion tree induced by `LegalOrientations`. Since the height of the recursion tree is $O(\log |V_M|)$, i.e. the edges of C, and the cost of each recursion node is $O(\log^2 |V_M|)$, delay between any two consecutive solutions is bounded by $O(\log^3 |V_M|)$. As it can be seen, it is crucial that the size of C is (poly)logarithmic. □

Lemma 12. *Regarding Algorithm 5, the following properties hold:*

1. *All the extended cyclic orientations of M are output.*
2. *Only extended cyclic orientations of M are output.*
3. *There are no duplicates.*

As a result, we obtain an algorithm with delay $\tilde{O}(|E_M|)$.

Lemma 13. *The extended cyclic orientations of $M(V_M, E_M)$ can be enumerated with delay $\tilde{O}(|E_M|)$ and space $O(|E_M|)$.*

Proof. Finding extended orientations $\overrightarrow{M'}$ of M' can be done with $O(|E'_M|)$ delay. Every time a new $\overrightarrow{M'}$ has been generated, we apply Algorithm 6. By Lemma 11 we output the first cyclic orientation $\overrightarrow{M}$ of M with delay $O(|E_M| \log |V_M|)$ and the remaining ones with delay $O(\log^3 |V_M|)$. Hence the maximum delay between any two consecutive solutions is $O(|E'_M| + |E_M| \log |V_M|) = O(|E_M| \log |V_M|) = \tilde{O}(|E_M|)$. The space usage is linear: in particular in Algorithm 6 the space is $O(\log^2 |V_M|)$, because of the reachability matrix R, which is smaller than $O(|E_M|)$. □

By Lemmas 13 and 9, and considering the setup cost in Section 6.1 ($|V_M| \leq |V|$ and $|E_M| \leq |E|$), we can conclude as follows.

Theorem 3. *Algorithm 5 lists all cyclic orientations of $G(V, E)$ with setup cost $O(|V|^2)$ and delay $\tilde{O}(|E|)$. The space usage is $O(|E|)$.*

It is possible to modify our approach to get a setup time equal to the delay, requiring space $\Theta(|V| \cdot |E|)$.

Theorem 4. *All cyclic orientations of $G(V, E)$ can be listed with setup cost $\tilde{O}(|E|)$, delay $\tilde{O}(|E|)$, and space usage of $\Theta(|V| \cdot |E|)$.*

Proof. We use $n = |V|$ and $m = |E|$ for brevity. Let A_1 be the following algorithm that takes $T_1 = O(mn)$ time to generate n solutions, each with $\tilde{O}(m)$ delay, starting from any given cycle of size $\geq \log n$. This cycle is found by performing a BFS on an arbitrary node u, and identifying the shortest cycle C_u containing u. Now, if $|C_u| < \log n$, since C_u is a log-hole as required, we stop the setup and run the algorithms in the previous sections setting $C = C_u$. The case of interest in this section is when $|C_u| \geq \log n$. We take a cyclic orientation $\overrightarrow{C_u}$ of C_u, and then n arbitrary orientations of the edges in $G \setminus C_u$. The setup cost is $O(m)$ time and we can easily output each solution in $\tilde{O}(m)$ delay. We denote this set of n solutions by Z_1.

Also, let A_2 be the algorithm behind Theorem 3, with a setup cost of $O(mn)$ and $\tilde{O}(m)$ delay (i.e. Algorithm 5). We denote the time taken by A_2 to list the first n solutions, including the $O(mn)$ setup cost, by $T_2 = \tilde{O}(mn)$, and this set of n solutions by Z_2. Since Z_1 and Z_2 can have nonempty intersection, we want to avoid duplicates.

We show how to obtain an algorithm A that lists all the cyclic orientations without duplicates with $\tilde{O}(m)$ setup cost and delay, using $O(mn)$ space. Even though the delay cost of A is larger than that of A_1 and A_2 by a constant factor, the asymptotic complexity is not affected by this constant, and remains $\tilde{O}(m)$.

Algorithm A executes simultaneously and independently the two algorithms A_1 and A_2. Recall that these two algorithms take $T_1 + T_2$ time in total to generate Z_1 and Z_2 with $\tilde{O}(m)$ delay. However those in Z_2 are produced after a setup cost of $O(mn)$. Hence A slows down on purpose by a constant factor c, thus requiring $c(T_1 + T_2)$ time: it has time to find the distinct solutions in $Z_1 \cup Z_2$ and build a dictionary D_1 on the solutions in Z_1. (Since an orientation can be represented as a binary string of length m, a binary trie can be employed as dictionary D_1, supporting each dictionary operation in $O(m)$ time.) During this time, A outputs the n solutions from Z_1 with a delay of $c(T_1 + T_2)/n = \tilde{O}(m)$ time each, while storing the rest of solutions of $Z_2 \setminus Z_1$ in a buffer Q.

After $c(T_1 + T_2)$ time, the situation is the following: Algorithm A has output the n solutions in Z_1 with $\tilde{O}(m)$ setup cost and delay. These solutions are stored in D_1, so we can check for duplicates. We have buffered at most n solutions of $Z_2 \setminus Z_1$ in Q. Now the purpose of A is to continue with algorithm A_2 alone, with $\tilde{O}(m)$ delay per solution, avoiding duplicates. Thus for each solution given by A_2, algorithm A suspends A_2 and waits so that each solution is output in $c(T_1 + T_2)/n$ time: if the solution is not in D_1, A outputs it; otherwise A extracts one solution from the buffer Q and outputs the latter instead. Note that if there are still d duplicates to handle in the future, then Q contains exactly d solutions from $Z_2 \setminus Z_1$ (and Q is empty when $A - 2$ completes its execution). Thus, A never has to wait for a non-duplicated solution. The delay is the maximum between $c(T_1 + T_2)/n$ and the delay of A_2, hence $\tilde{O}(m)$. The additional space is dominated by that of Q, namely, $O(mn)$ space to store up to n solutions. □

We also have a bound of $\tilde{O}(|E_M|)$ on the amortized cost using the lemma below with $f(x) = \tilde{O}(x)$ and $s = |V|$.

Lemma 14. *Listing all the extended cyclic orientations of $M(V_M, E_M)$ with delay $O(f(|E_M|))$ and setup cost $O(s \cdot |V_M|)$ implies that the average cost per solution is $O(f(|E_M|) + |E_M|)$.*

Proof. We perform a BFS on an arbitrary node u, and identify the shortest cycle $C_u(V_u, E_u)$ that contains u. This costs $O(m)$ time. Note that $C_u(V_u, E_u)$ is a hole (i.e. it has no chords). Note that a minimum cycle in M either is C_u or contains a node in $V_M - V_u$: hence we perform all the BFSs from each node in $V_M - V_u$, as explained in [15] with an overall cost of $O(|V_M| \cdot |V_M - V_u|)$. The number of extended orientations of M is at least $2^{|E_M - E_u|} \geq 2^{|V_M - V_u|}$. Our setup cost is $O(s \cdot |V_M|)$, with $s \leq |V_M|$, and the number of solutions is at least 2^s. The overall average cost per solution is at most $O(2^s \cdot f(|E_M|) + s \cdot |V_M|)/2^s$, which is $O\left(f(|E_M|) + |E_M| \cdot \frac{s}{2^s}\right)$. □

7. Single Source Cyclic Orientations (SSCO)

For the sake of completeness, we show how the techniques presented can be used to enumerate all the cyclic orientations of an undirected graph $G(V, E)$, whose only source is a given node s. Our algorithm is similar to that in Section 6 except for the usage of extended orientations, and works as follows.

1. Find a chordless cycle of small size $C(V_C, E_C)$ in $G(V, E) \setminus \{s\}$, called *hole* , and remove E_C from E, obtaining $G'(V, E')$, where $E' = E \setminus E_C$.
2. Find all the orientations $\overrightarrow{G'}$ of G', where s is a source and the nodes in V_C can be sources. We call these orientations *special orientations*, which can be listed with linear delay.
3. For each special orientation $\overrightarrow{G'}$, exploit all the possible ways of orienting the edges in E_C, putting them in $\overrightarrow{G'}$ in a way that the only source is s and there is at least a cycle.

The three steps above are summarized in Algorithm 7 and explained more in detail respectively in Sections 7.1–7.3.

7.1. Finding and removing a cycle not involving s

Given the graph G in input, we find a hole $C(V_C, E_C)$ not involving s in G (in linear time with a simple DFS after removing s) and we remove the edges in E_C from G, obtaining $G'(V, E')$, where $E' = E \setminus E_C$. We will denote the length of C as h, that is $|V_C| = h$. Note that either C exists, or the graph does not allow any single source cyclic orientation.

Algorithm 7: Find the cyclic orientations of G with single source s

Input: An undirected connected graph $G(V, E)$
Output: All the cyclic orientations $\overrightarrow{G}(V, \overrightarrow{E})$

Find and remove a cycle not involving s (Section 7.1):
$C(V_C, E_C) \leftarrow$ any hole of $G \setminus \{s\}$ (i.e. a chordless cycle not involving s)
$G'(V, E') \leftarrow$ delete C's edges from G (i.e. $E' = E \setminus E_C$)

Enumerate special orientations (Section 7.2) and put back the hole (Section 7.3):
for each special orientation $\overrightarrow{G}'$ of G' **do**
  for each legal orientation $\overrightarrow{C}$ of C (see Algorithm 8) **do**
    $\overrightarrow{G}(V, \overrightarrow{E}'') \leftarrow$ combine $\overrightarrow{G}'$ and $\overrightarrow{C}$, where $\overrightarrow{E}'' = \overrightarrow{E}' \cup \overrightarrow{E}_C$
    Output $\overrightarrow{G}$

Algorithm 8: Returning all legal orientations of C

Input: $\overrightarrow{G'}(V, \overrightarrow{E'})$ acyclic, a cycle $C(V_C, E_C)$ with $V_C \subseteq V$
Output: All the legal orientations $\overrightarrow{C}(V_C, \overrightarrow{E}_C)$

Build the reachability matrix R for the nodes of V_C in $\overrightarrow{G'}$
Let $V_C = \{c_1, \ldots c_h\}$, where $c_{h+1} \equiv c_1$ by definition and c_1 has indegree > 0 in G'.
Execute `LegalOrientations` $(\overrightarrow{C}'(\emptyset, \emptyset), 1, R)$

Procedure `LegalOrientations`$(\overrightarrow{C}'(V'_C, \overrightarrow{E}'_C), j, R)$
  if $j = h + 1$ **then**
    output $\overrightarrow{C}'$
  else
    $u \leftarrow c_j$, $v \leftarrow c_{j+1}$
    $R_1 \leftarrow R$ updated by adding the arc (u, v)
    if (R_1 *is cyclic or has positive reachability test on* $\{c_{j+1}, \ldots, c_{h+1}\}$) *and* ($u$ *is not a source in* $\overrightarrow{G}''(V, \overrightarrow{E'} \cup \overrightarrow{E}'_C \cup \{(u, v)\})$) **then**
      `LegalOrientations` $(\overrightarrow{C}'(V'_C, \overrightarrow{E}'_C \cup \{(u, v)\}), j + 1, R_1)$
    $R_2 \leftarrow R$ updated adding the arc (v, u)
    if (R_2 *is cyclic or has positive reachability test on* $\{c_{j+1}, \ldots, c_{h+1}\}$) *and* ($v$ *is not a source in* $\overrightarrow{G}''(V, \overrightarrow{E'} \cup \overrightarrow{E}'_C \cup \{(v, u)\})$) **then**
      `LegalOrientations` $(\overrightarrow{C}'(V'_C, \overrightarrow{E}'_C \cup \{(v, u)\}), j + 1, R_2)$

7.2. Enumerating special orientations

Given G', the node s and the nodes V_C, we want all the orientations such that s is a source and nodes in V_C can be sources. This can be trivially done modifying the algorithm presented in Section 3. In particular, by redefining the notion of *valid direction assignment* as follows.

Definition 8 (*Valid Direction Assignment*)**.** Given $\overrightarrow{G}_{i-1}(V_{\leq i-1}, \overrightarrow{E})$, the direction assignment $\overrightarrow{Z}_i$ is *valid* if

- any $v_j \neq s$ and not in V_C that is *full* in $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{Z}_i)$ is *not* a source, for $1 \leq j \leq i$.

The definition of full node, i.e. Definition 1, as well the algorithm remains the one of Section 3. The fact that s is a source in the final orientation is still guaranteed by the fact that we are using the good order in Definition 3.

The idea behind the new definition of valid direction assignment is the following. When dealing with $\overrightarrow{G}_i$, since full nodes sources in $\overrightarrow{G}_i$ will be sources also in any extension of $\overrightarrow{G}_i$, i.e. in the final orientation $\overrightarrow{G}$, we impose that the only nodes that can be full sources are the ones in V_C.

Similarly to Section 3, the following result can be easily proved.

Lemma 15. *Special orientations can be listed with linear delay.*

It is worth observing that the notion of special orientation here replaces the notion of extended orientation in Section 6.3.

7.3. Putting back the hole

As for putting back the hole in Step (3), for each listed $\overrightarrow{G'}$ we have to decide how to put back the edges of the cycle C, namely, how to find the orientations of C that create directed cycles with s being the only source. This phase is similar to the one in Section 6.4, but it deals with simple orientations of C rather than extended orientations.

Considering this difference, the corresponding definition of *legal orientation* is the following.

Definition 9. Given the cycle $C(V_C, E_C)$ and $\overrightarrow{G}'(V, \overrightarrow{E}')$, we call *legal orientation* $\overrightarrow{C}(V_C, \overrightarrow{E}_C)$ any orientation of C such that the resulting graph $\overrightarrow{G}''(V, \overrightarrow{E}'')$, where $\overrightarrow{E}'' = \overrightarrow{E}' \cup \overrightarrow{E}_C$, is cyclic and s is the only source.

In particular, we have to efficiently deal with the following problem: given a cycle $C(V_C, E_C)$ and given a special orientation $\overrightarrow{G'}$, enumerate all the legal orientations $\overrightarrow{C}(V_C, \overrightarrow{E}_C)$ of C. We remark that for each G' we have at least two legal orientations, namely the clockwise and the counter-clockwise orientations of C. In order to also enumerate the other legal orientations of C, as in Section 6.4, we exploit a reachability matrix R for the nodes in V_C in $\overrightarrow{G'}$. This matrix can be built with a time cost $O(|E'| \cdot h)$. All the steps are shown by Algorithm 8.

Note that deciding the orientation of the edges in E_C is done with a binary partition instead of a ternary partition as in Algorithm 6, since we are dealing with simple orientations of C rather than extended orientations. Hence, for each edge $\{u, v\}$ in E_C, we try the two possible directions and update the reachability matrix R to check whether the current partial direction assignment will produce at least one solution. It is worth observing that both the update of R and the dead-end check can be done in $O(h^2)$ (that is, the size of R).

Analogously to Lemma 11, the following result holds.

Lemma 16. *Algorithm 8 outputs in $O(|E'| \cdot h + h^3)$ time the first legal orientation of C, and each of the remaining ones with $O(h^3)$ delay.*

As a result, we obtain the following lemma, whose proof is similar to the one of Lemma 13.

Lemma 17. *Given a cycle of length h in G without s, the cyclic orientations of $G(V, E)$ whose only source is s can be enumerated with delay $O(|E| \cdot h + h^3)$ and space $O(|E| + h^2)$.*

Let h be the girth of G without s; considering that we can apply the algorithm in [15] to find a cycle of length $\leq h + 1$ in G without s with cost $O(|V|^2)$, by using Algorithm 7 and applying Lemma 17 we can prove Theorem 5.

Theorem 5. *All cyclic orientations of $G(V, E)$ whose only source is s can be enumerated with setup cost $O(|V|^2 + |E| \cdot h + h^3)$, delay $O(|E| \cdot h + h^3)$, where h is the girth of $G \setminus \{s\}$, and space $O(|E| + h^2)$.*

We observe that the strategy described in Theorem 4 could be also applied to reduce the setup time, at the cost of increasing space usage, in Theorem 5.

8. Variations of the enumeration problems

We consider here some variations of SSAO that could be of independent interest for a given undirected connected graph $G(V, E)$ with n nodes and m edges.

single source acyclic orientations (weak SSAO): Given a set of nodes $S \subseteq V$, enumerate all the acyclic orientations $\overrightarrow{G}$ of G such that there is exactly one source x and $x \in S$.

multiple source acyclic orientations (strong MSAO): Given a set of nodes $S \subseteq V$, enumerate all the acyclic orientations $\overrightarrow{G}$ of G such that *all* the nodes in S are the only sources.[5]

multiple source acyclic orientations (weak MSAO): Given a set of nodes $S \subseteq V$, enumerate all the acyclic orientations $\overrightarrow{G}$ of G such that if x is a source then $x \in S$.[6]

We show that these problems can be reduced to SSAO. It is easy to see that weak SSAO can be solved simply by enumerating all the SSAOs in G with source s for each $s \in S$. It is worth observing that for each s there is at least a solution, meaning that the size of S does not influence the delay of weak SSAO.

Let us consider weak MSAO. To solve it, we create a dummy node s, and connect it to every node in S. More formally, we build $G'(V \cup \{s\}, E \cup E_s)$, where $E_s = \{\{s, x\} : x \in S\}$. Any weak MSAO of G can be transformed into a SSAO of G' if we add s and all edges in E_s (oriented away from s): s is a source and all nodes in S are no longer sources since they can be reached from s, hence s is the single source. Note that the orientation is still acyclic as s is a source and cannot be part of a cycle. The opposite is true as well: any SSAO of G' can be transformed into a weak MSAO of G by removing s and the edges in E_s. This process only removes edges incident to nodes in S, hence only nodes in S possibly become sources. Clearly the orientation is still acyclic as removing nodes and edges cannot create cycles.

Finally, consider strong MSAO. To solve it, we simply collapse all nodes of S into one node s. More formally, we generate $G''(V \cup \{s\} \setminus S, E \cup E_s)$, where $E_s = \{\{s, x\} : \exists y \in S \text{ with } \{y, x\} \in E\}$. As s and all nodes in S must be sources, all of their incident

[5] These orientations are possible if and only if S is an independent set.

[6] Not all nodes in S must be sources, but there cannot be sources in $V \setminus S$.

edges must be oriented away from them in all acyclic orientations, while the rest of the graph is exactly the same for both cases. Clearly, any SSAO for G'' induces a strong MSAO of G that can be obtained by removing s and E_s and re-integrating S and the edges between S and $V \setminus S$ (oriented away from S). Similarly, removing S (and the edges between S and $V \setminus S$) and integrating s and E_s (with edges oriented away from s), creates a SSAO for G'': there is an edge from s to any node in $V \setminus S$ that was previously connected with S, hence these nodes cannot be sources; all other nodes in $V \setminus S$ were not connected to S and hence their in-degrees and out-degrees are unchanged.

By Theorem 1, observing that the above transformations requires $O(m)$ time, we can conclude the following result.

Corollary 1. *Problems weak* SSAO*, strong* MSAO*, and weak* MSAO *can be solved with delay* $O(m \cdot n)$ *and space* $O(m)$.

Along the same lines, we can easily solve these variations.

single source cyclic (weak SSCO): Given a set of nodes $S \subseteq V$, enumerate all the cyclic orientations $\overrightarrow{G}$ of G such that there is exactly one source x and $x \in S$.

multiple source cyclic (strong MSCO): Given a set of nodes $S \subseteq V$, enumerate all the cyclic orientations $\overrightarrow{G}$ of G such that *all* the nodes in S are the only sources.[7]

multiple source cyclic (weak MSCO): Given a set of nodes $S \subseteq V$, enumerate all the cyclic orientations $\overrightarrow{G}$ of G such that if x is a source then $x \in S$.[8]

9. Conclusions

In this paper we have shown a collection of algorithms for enumerating graph orientations with constraints regarding sources and cycles. For the problem of AO we improve previous work by reducing the delay to $O(m)$; for all the other problems considered (namely SSO, SSAO, SSCO and CO) we show the first algorithms with guaranteed delay. The algorithms also solve some generalizations, as discussed in Section 8 but the problem posed by Squire [24], of efficiently enumerating acyclic orientations with sources and targets lying within two arbitrary sets S and T, remains open.

Acknowledgments

We wish to thank the anonymous referees for their valuable comments. This work has been partially supported by the Italian Ministry of Education, University, and Research (MIUR), grant identifier PRIN 2012C4E3KT national research project AMANDA — Algorithmics for MAssive and Networked DAta.

References

[1] N. Alon, M. Tarsi, Colorings and orientations of graphs, Combinatorica 12 (2) (1992) 125–134.
[2] N. Alon, Z. Tuza, The acyclic orientation game on random graphs, Random Structures Algorithms 6 (2–3) (1995) 261–268.
[3] D. Avis, K. Fukuda, Reverse search for enumeration, Discrete Appl. Math. 65 (1) (1996) 21–46.
[4] V.C. Barbosa, J.L. Szwarcfiter, Generating all the acyclic orientations of an undirected graph, Inform. Process. Lett. 72 (1) (1999) 71–74.
[5] B. Benson, D. Chakrabarty, P. Tetali, G-parking functions, acyclic orientations and spanning trees, Discrete Math. 310 (8) (2010) 1340–1353.
[6] B. Bollobas, Extremal Graph Theory, Dover Publications, Incorporated, 2004.
[7] A. Conte, R. Grossi, A. Marino, R. Rizzi, Enumerating cyclic orientations of a graph, in: Combinatorial Algorithms - 26th International Workshop, IWOCA 2015, Verona, Italy, October 5–7, 2015, Revised Selected Papers, 2015, pp. 88–99.
[8] A. Conte, R. Grossi, A. Marino, R. Rizzi, Listing acyclic orientations of graphs with single and multiple sources, in: LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, April 11–15, 2016, Proceedings, 2016, pp. 319–333.
[9] A. Conte, R. Grossi, A. Marino, R. Rizzi, L. Versari, Directing road networks by listing strong orientations, in: International Workshop on Combinatorial Algorithms, Springer, 2016, pp. 83–95.
[10] A. Conte, R. Grossi, A. Marino, L. Versari, Sublinear-space bounded-delay enumeration for massive network analytics: Maximal cliques, in: 43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11–15, 2016, Rome, Italy, 2016, 148:1–148:15.
[11] P. Erdős, G. Katona, B.J.M. Társulat, Theory of Graphs: Proceedings of the Colloquium Held at Tihany, Hungary, September 1966, Academic Press, 1968.
[12] P. Erdős, L. Pósa, On the maximal number of disjoint circuits of a graph, Publ. Math. Debrecen 9 (1962) 3–12.
[13] D.C. Fisher, K. Fraughnaugh, L. Langley, D.B. West, The number of dependent arcs in an acyclic orientation, J. Combin. Theory Ser. B 71 (1) (1997) 73–78.
[14] B. Iriarte, Graph orientations and linear extensions, in: DMTCS Proc., 2014, pp. 945–956.
[15] A. Itai, M. Rodeh, Finding a minimum circuit in a graph, SIAM J. Comput. 7 (4) (1978) 413–423.
[16] R. Johnson, Network reliability and acyclic orientations, Networks 14 (4) (1984) 489–505.
[17] D.S. Johnson, C.H. Papadimitriou, M. Yannakakis, On generating all maximal independent sets, Inf. Process. Lett. 27 (3) (1988) 119–123.
[18] N. Linial, Hard enumeration problems in geometry and combinatorics, SIAM J. Algebr. Discrete Methods 7 (2) (1986) 331–335.
[19] J. Moon, Topics on Tournaments, in: Athena Series: Selected Topics in Mathematics, Holt, Rinehart and Winston, 1968.
[20] O. Pikhurko, Finding an unknown acyclic orientation of a given graph, Combin. Probab. Comput. 19 (2010) 121–131.
[21] B. Roy, Nombre chromatique et plus longs chemins d'un graphe, Rev. Fr. Autom. Inform. Rech. Oper. 1 (5) (1967) 129–132.
[22] B. Schwikowski, E. Speckenmeyer, On enumerating all minimal solutions of feedback problems, Discrete Appl. Math. 117 (1) (2002) 253–265.
[23] A. Setiawan, S.-i. Nakano, Listing all st-orientations, IEICE Trans. Fundam. Electron. Commun. Comput. Sci. 94 (10) (2011) 1965–1970.

[7] These acyclic orientations are possible if and only if S is an independent set.

[8] Not all nodes in S must be sources, but there cannot be sources in $V \setminus S$.

[24] M.B. Squire, Generating the acyclic orientations of a graph, J. Algorithms 26 (2) (1998) 275–290.
[25] R.P. Stanley, Acyclic orientations of graphs, Discrete Math. 5 (2) (1973) 171–178.
[26] R.P. Stanley, What Is Enumerative Combinatorics? Springer, 1986.
[27] J.L. Szwarcfiter, R.C. Persiano, A.A. Oliveira, Orientations with single source and sink, Discrete Appl. Math. 10 (3) (1985) 313–321.
[28] R.E. Tarjan, A note on finding the bridges of a graph, Inform. Process. Lett. 2 (6) (1974) 160–161.
[29] L. Vitaver, Determination of minimal coloring of vertices of a graph by means of Boolean powers of the incidence matrix, Dokl. Akad. Nauk SSSR 147 (1962) 728.