# Listing Acyclic Orientations of Graphs with Single and Multiple Sources

Alessio Conte, Roberto Grossi, Andrea Marino, Romeo Rizzi

# Listing Acyclic Orientations of Graphs with Single and Multiple Sources

Alessio Conte[1], Roberto Grossi[1], Andrea Marino[1], and Romeo Rizzi[2]

[1] Università di Pisa and Erable, Inria, conte,grossi,marino@di.unipi.it*
[2] Università di Verona, rizzi@di.univr.it

**Abstract.** We study enumeration problems for the acyclic orientations of an undirected graph with $n$ nodes and $m$ edges, where each edge must be assigned a direction so that the resulting directed graph is acyclic. When the acyclic orientations have single or multiple sources specified as input along with the graph, our algorithm is the first one to provide guaranteed bounds, giving new bounds with a delay of $O(m \cdot n)$ time per solution and $O(n^2)$ working space. When no sources are specified, our algorithm improves over previous work by reducing the delay to $O(m)$, and is the first one with linear delay.

## 1 Introduction

Acyclic orientations of graphs are related to several basic problems in graph theory. An *orientation* of an undirected graph $G$ is the directed graph $\overrightarrow{G}$ whose arcs are obtained assigning a direction to each edge in $G$. The orientation $\overrightarrow{G}$ is *acyclic* when it does not contain cycles, and a node $s$ is a *source* in $\overrightarrow{G}$ if it has indegree zero. For instance, consider the graphs in Figure 1. The directed graph in (b) is an acyclic orientation for the undirected graph in (a). In particular, since the orientation has only one source ($v_9$), it is called a *single source acyclic orientation*.

Starting from the observation that each acyclic orientation corresponds to a partial order for the underlying graph, Iriarte [7] investigates which orientations maximize the number of linear extensions of the corresponding poset. Alon and Tarsi [1] look for special orientations to give bounds on the size of the maximum independent set or the chromatic number. Gallai, Roy, and Vitaver independently describe a well-known result stating that every orientation of a graph with chromatic number $k$ contains a simple directed path with $k$ vertices [6, 12, 17]. There are further problems that can be addressed by looking at acyclic orientations. For instance, Benson et al. [4] show 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.

* This work has been partially supported by the Italian Ministry of Education, University, and Research (MIUR) under PRIN 2012C4E3KT national research project AMANDA — Algorithmics for MAssive and Networked DAta.

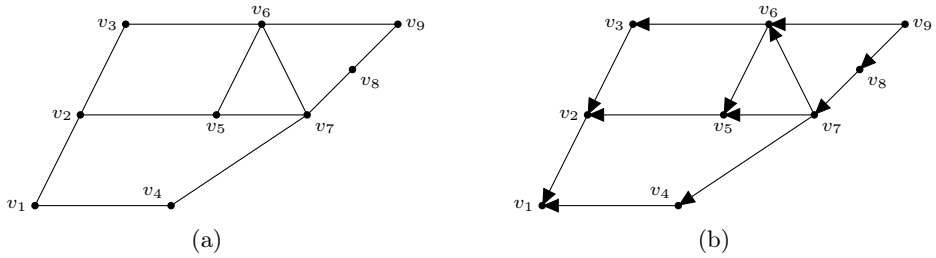


**Fig. 1.** An undirected connected graph without self-loops (a) and one of its acyclic orientations (b).

Counting how many acyclic orientations can be found in a graph is a fundamental problem in combinatorics, dating back to the 70s or earlier [16]. Linial [10] proves that this problem is #P-complete. Stanley [15] shows how the number of acyclic orientations can be computed by using the chromatic polynomial (a special case of Tutte's polynomial). Another approach that concerns the number of acyclic orientations is the acyclic orientation game. Alon and Tuza [2] inquire about the amount of oriented edges needed to define a unique orientation of $G$, and find this number to be almost surely $\Theta(|V|\log|V|)$ in Erdős–Rényi random graphs. Pikhurko [11] shows that the number of these edges in the worst case is no greater than $(\frac{1}{4}+o(1))|V|^2$ for general graphs.

**Problems addressed.** Our paper investigates new algorithms for enumerating patterns for this interesting problem in graph theory, given an undirected connected graph $G(V,E)$ with $n$ nodes and $m$ edges.

**single source acyclic orientations (SSAO):** Given a node $s \in V$, enumerate all the acyclic orientations $\overrightarrow{G}$ of $G$, such that $s$ is the only source.

**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.[3]

**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 $x \in S$.[4]

**acyclic orientations (AO):** Enumerate all the acyclic orientations $\overrightarrow{G}$ of $G$.

We will show that these problems can be reduced to SSAO, with a one-to-one correspondence between their solutions. Many other variants with constraints on the number or choice of sources can be reduced to SSAO as well: the ones

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

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

we present are some of the most representative ones. We analyze the cost of an enumeration algorithm for SSAO, weak SSAO, strong MSAO, weak MSAO, and AO in terms of its *delay* cost, 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. [8]). We are interested in algorithms with guaranteed delay and space.

**Previous work.** We are not aware of any provably good bounds for problems SSAO, weak SSAO, strong MSAO and weak MSAO. Johnson's backtracking algorithm [9] for SSAO has been presented over 30 years ago to solve problems on network reliability. However its complexity is not given and is hard to estimate, as it is based on a backtracking approach with dead ends.

In his paper presenting an algorithm for AO, Squire [14] writes that he has been unable to efficiently implement Johnson's approach because of its dead ends. Squire's algorithm for AO uses Gray codes and has an amortized cost of $O(n)$ per solution, but its delay can be $O(n^3)$ time for a solution. The algorithm by Barbosa and Szwarcfiter [3] solves AO with an amortized time complexity of $O(n+m)$ per solution, delay $O(n \cdot m)$. The algorithm builds the oriented graph incrementally by iteratively adding the nodes to an empty directed graph.

It is worth observing that by replacing each edge with a double arc and applying any algorithm for maximal feedback arc set enumeration, one can obtain all the acyclic orientations of $G$. State of the art approaches for the latter problem guarantee a delay $\Omega(n^3)$ as shown by Schwikowski and Speckenmeyer [13].

All the techniques above for AO, including the one in [14], do not to extend smoothly to SSAO, weak SSAO, strong MSAO, and weak MSAO. In a previous work [5], we studied the related problem of enumerating the cyclic orientations of an undirected graph, but the proposed techniques cannot be reused for the problems in this paper.

**Our results.** Our contribution is the design of the first enumeration algorithms with guaranteed bounds for SSAO, weak SSAO, strong MSAO, and weak MSAO: the complexity is $O(m \cdot n)$ delay per solution using $O(n^2)$ space. For AO, we also show how to obtain $O(m)$ delay, improving the delay of [3, 13, 14], but we do not improve the amortized cost of $O(n)$ in [14].

We therefore focus on SSAO in the first part of the paper, and then show an optimization that holds for the case of AO. We guarantee that, at any given partial solution, the extensions of the partial solution will enumerate new acyclic orientations. To this aim, we solve several non-trivial issues.

- We use a recursive approach where each call surely leads to a solution. For SSAO this is achieved also by using a suitable ordering of the nodes.
- We quickly identify the next recursion calls within the claimed time delay.
- We do a careful analysis of the recursion tree, and show how to check efficiently for node reachability during recursion.
- In the case of AO we exploit the fact that the recursion tree does not contain unary nodes.

The paper is organized as follows. We give the necessary definitions and terminology in Section 2. We then discuss how to solve SSAO in Section 3 and further reduce the delay for AO in Section 4. In Section 5 we show how weak SSAO, strong MSAO, weak MSAO, and AO reduce to SSAO. We draw some conclusions in Section 6.

## 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. 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 wlog 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 ordering of the nodes of $G$. We define $V_{\leq i}$ as the set $\{v_1, \ldots, v_i\}$, $N(v_i) = \{x : \{v_i, x\} \in E\}$ as the set of neighbors of the node $v_i$, and $N_{\leq i}(v)$ as the set $N(v) \cap V_{\leq i}$. For brevity, $N_<(v_j)$ means $N_{\leq j-1}(v_j)$. Clearly we have $\sum_{j=1}^{n} |N_<(v_j)| = m$.

Starting from an empty directed graph, for increasing values of $i = 1, 2, \ldots, n$, our algorithms add $v_i$ to the current graph and recursively exploit all the possible ways of directing the edges $\{v_i, x\}$ with $x \in N_<(v_i)$, called direction assignments: a *direction assignment* $\overrightarrow{Z}$ for $v_i$ is an orientation of the set of edges $\{\{v_i, x\} : x \in N_<(v_i)\}$. We refer to the following special assignments as

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

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

We denote by $\overrightarrow{G}_0$ the starting empty directed graph, and by $\overrightarrow{G}_i$ the graph whose last added node is $v_i$, with $1 \leq i \leq n$.

## 3 Single Source Acyclic Orientations (SSAO)

Given a graph $G$ and a node $s$, this section describes how to enumerate its acyclic orientations $\overrightarrow{G}$ such that $s$ is the unique source in $\overrightarrow{G}$. Starting from an empty graph $\overrightarrow{G}_0$, our algorithm adds $v_i$ to $\overrightarrow{G}_{i-1}$ for $i = 1, 2, \ldots, n$. For the edges in $N_<(v_i)$, it exploits all the suitable direction assignments $\overrightarrow{Z}$: each assignment gives a certain $\overrightarrow{G}_i$, on which it recurses. Every time it adds the last vertex $v_n$ in a recursive call, it outputs the corresponding $\overrightarrow{G}_n$ as a new solution $\overrightarrow{G}$.

The above simple scheme can lead to dead ends in its recursive calls, where partial orientations $\overrightarrow{G}_i$ cannot be extended to reach $\overrightarrow{G}_n$, i.e. an acyclic $\overrightarrow{G}$ whose only source is $s$. We prevent this situation by examining the nodes of $G$ in a suitable order that allows us to exploit the following notions.

**Definition 1 (full node).** *Given an ordering $v_1, \ldots, v_n$ of the nodes in $G$, a node $v_j$ ($1 \leq j \leq i$) is* full *in $\overrightarrow{G}_i$ if $N_{\leq i}(v_j) = N(v_j)$.*

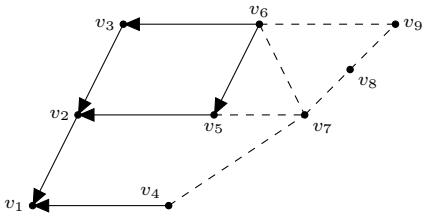


**Fig. 2.** A partial acyclic orientation (thick edges). To add $v_7$ to the partial orientation we exploit valid direction assignments for undirected edges $\{v_4, v_7\}, \{v_5, v_7\}, \{v_6, v_7\}$.

**Definition 2 (valid direction assignment).** *Given* $\overrightarrow{G}_{i-1}(V_{\leq i-1}, \overrightarrow{E})$*, the direction assignment* $\overrightarrow{Z}$ *is* valid *if*

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

The rationale is the following. When dealing with $\overrightarrow{G}_i$, full nodes are the ones whose edges in $G$ have been all already assigned in $\overrightarrow{G}_i$. This means that if a full node is a source in $\overrightarrow{G}_i$ it will be a source also in any extension of $\overrightarrow{G}_i$, i.e. in the final orientation $\overrightarrow{G}$. A valid direction assignment imposes that we do not create cycles and each full node (except $s$) is not a source. We will deal with orientations of $\overrightarrow{G}_i$ that are the outcome of a sequence of valid direction assignments: such orientations will be referred to as *partial acyclic orientations* of $G_i$.

Consider the graph in Figure 1 (a). We we want acyclic orientations whose only source is $s = v_9$ processing the nodes in the order $v_1, \ldots, v_9$. In particular, consider the situation in Figure 2. We have the graph $\overrightarrow{G}_6$ (thick edges) and we have to add the vertex $v_7$, deciding how to orient the edges to the vertices in $N_<(v_i)$, namely $\{v_4, v_7\}, \{v_5, v_7\}, \{v_6, v_7\}$, to obtain all the possible $\overrightarrow{G}_7$. Notice that $v_1$, $v_2$, and $v_3$ are full in $\overrightarrow{G}_6$ since the edges in their whole neighborhood have been already oriented: hence since they are not source in $\overrightarrow{G}_6$, they will be not sources in the final orientation. The direction assignment $\{(v_7, v_4), (v_5, v_7), (v_6, v_7)\}$ is valid since the full nodes in the corresponding $\overrightarrow{G}_7$, namely $v_1, v_2, v_3, v_4$ are not sources, and $\overrightarrow{G}_7$ is acyclic. On the other hand, $\{(v_4, v_7), (v_5, v_7), (v_6, v_7)\}$ is not a valid direction assignment since $v_4$ is a source full in the corresponding $\overrightarrow{G}_7$ and it will be source in any final orientation extending $\overrightarrow{G}_7$. Notice that, since $v_7$, and $v_9$ are not in $N_<(v_6)$, $v_6$ can be source in $\overrightarrow{G}_6$ and can remain source also in $\overrightarrow{G}_7$, while it should be not a source in $\overrightarrow{G}_9$. Hence, exploiting just the valid direction assignments means taking care that no cycle is created and no full node is source.

In order to efficiently find valid direction assignments, our algorithm uses an ordering of the nodes $v_1, \ldots, v_n$ that satisfies the conditions below.

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

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

**Algorithm 1:** `single-source-acyclic`

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

**if** $i > n$ **then** output $\overrightarrow{G}$; **return** ;
Execute Algorithm 2;
**for** any valid direction assignment $\overrightarrow{Z}$ for $v_i$ starting from $\overrightarrow{Z} = Y_i$ **do**
  `single-source-acyclic`$(G, \overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{Z}), i + 1)$;

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 $v_i$, when adding the latter to $\overrightarrow{G}_{i-1}$. Dead ends can be avoided in this way: when adding $v_i$ to $\overrightarrow{G}_{i-1}$, we have at least one solution extending $\overrightarrow{G}_{i-1}$ in which $v_i$ is not a source. Indeed the following property holds.

*Property 1.* 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.* For any $j > i$, consider the valid direction assignment $Y_j$, i.e. $\{(v_j, x) : x \in N_<(v_j)\}$ obtaining $\overrightarrow{G}$. These direction assignments cannot create cycles in $\overrightarrow{G}_j$ and the only final source is $s$. □

Referring to the graph in Figure 1 (a), if $s = v_9$, the order induced by $v_1, \ldots, v_9$ is a good order. Considering $\overrightarrow{G}_6$ in Figure 2, according to Property 1, $Y_7$ corresponds to $\{(v_7, v_6), (v_7, v_5), (v_7, v_4)\}$, $Y_8$ corresponds to $\{(v_8, v_7)\}$, while $Y_9$ corresponds to $\{(v_9, v_8), (v_9, v_6)\}$. These direction assignments are valid and lead to the acyclic orientation in Figure 1 (b) whose only source is indeed $v_9$.

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: node $s$ is the last node and, for each node, its parent in the DFS tree appears after it in the order.

Our recursive algorithm is shown in Algorithm 1, where the good ordering of the nodes is employed. The initial call is `single-source-acyclic`$(G, \overrightarrow{G}_0, 1)$. The algorithm recursively exploits 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. The latter assignments are generated by a recursive computation. The general picture of our solution can be seen as follows: we have the primary recursion tree to generate all the wanted cyclic orientations (Algorithm 1), where each node has associated a secondary recursion tree to generate locally all the valid direction assignments (Algorithm 2). A naive implementation would simply consider each of the $2^{|N_<(v_i)|}$ direction assignments checking whether it is valid or not (e.g. using a DFS in $O(m)$ time). Instead, the following section introduces an efficient method that allows us to iterate through valid direction assignments only.

**Algorithm 2:** Returning valid direction assignments

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

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

**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` $(G, \overrightarrow{G}_{i-1}, v_i, \overrightarrow{W} \cup \{(x_j, v_i)\}, j+1, R, B)$;

### 3.1 Iterating over valid direction assignments

Given $\overrightarrow{G}_{i-1}$ and node $v_i$, we show how to iterate over valid direction assignments: our approach is shown in Algorithm 2. By definition, each valid direction assignment $\overrightarrow{W}$ of the edges in $\{\{v_i, x\} : x \in N_<(v_i)\}$ should guarantee that we are not creating a cycle and no new full node becomes a source.

Let $F_i$ be the set of nodes which are: (a) full in $\overrightarrow{G}_i$, (b) sources in $\overrightarrow{G}_{i-1}$, (c) not full in $\overrightarrow{G}_{i-1}$. Note that $F_i \subseteq N_<(v_i)$. 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 $\{(v_i, x) : x \in F_i\}$ to $\overrightarrow{W}$. Observe that this is mandatory for the orientations of the corresponding edges because otherwise a node in $F_i$ would become a source in the final orientation $\overrightarrow{G}$. Also, $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{W})$ is acyclic.

After that, we have to decide the orientation of the remaining edges. This part relies on procedure `Generate` in Algorithm 2. In particular, we have to assign a direction to the edges $\{v_i, u\}$ for each node $u$ in $N_<(v_i) \setminus F_i$ and check if they do not create cycles. We take these nodes in arbitrary order as $x_1, \ldots, x_k$ and, for increasing values of $j = 1, 2, \ldots, k$, we do the following: if the arc $e \in \{(v_i, x_j), (x_j, v_i)\}$ does not create a cycle in $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{W})$, proceed recursively with $\overrightarrow{W} = \overrightarrow{W} \cup \{e\}$.

For a given $v_i$, the reachability tests above for $x_j$ $(j = 1, 2, \ldots, k)$ can be performed with $O(k)$ (forward and backward) DFS traversals, requiring overall $O(k \cdot m)$ time. Since $k$ is bounded by the degree of $v_i$ and can be $O(n)$ in the worst case, we propose a solution that reduces the cost from $O(k \cdot m)$ to $O(m)$ time. It truncates the DFSes using sets $B$ and $R$ to avoid visiting the nodes in these sets. Since the partially built 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 coloring the nodes of $B$ and $R$ showing that the overall time required by these tests for each valid direction assignment is $O(m)$.

**Lemma 1.** *Algorithm 2 returns valid direction assignments with delay $O(m)$.*

*Proof.* The arcs directed to nodes in $F_i$ are unchanged for all the valid direction assignments for $v_i$ and can be computed in $O(m)$ time at the beginning.

When exploring the possible orientations of edges $\{v_i, x_j\}$, for $j = 1, 2, \ldots, k$, each time we have to decide whether $(v_i, x_j)$ or $(x_j, v_i)$ creates a cycle or not when added to $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{W})$. To this aim we color incrementally the nodes: 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*. Initially, all the nodes are uncolored, are $R$ and $B$ are empty. Since $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{W})$ is acyclic any node has just one color or is uncolored.

We now show that the sum of the costs to update the colors to produce a solution (valid direction assignment) 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: the leaf-to-root path of the current solution and the root-to-next-leaf path for the next solution (actually only the latter for the first solution). Observe that the former cost is always $O(|N_<(v_i)|)$. We prove that the sum of the costs from the root to a leaf in the secondary recursion tree induced by `Generate` is bounded by $O(m)$. 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 $x_j$ would be black also and thus $\overrightarrow{G}_i$ cyclic). On the other hand, when adding the arc $(x_j, v_i)$ to $\overrightarrow{W}$ we have only to make the traversed uncolored nodes black: once again, this corresponds to continue 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 2 recognizes that there are no more valid direction assignments, using time $O(m)$.

**Lemma 2.** *Referring to Algorithm 1, 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.* We prove the three statements separately.

1. Given a good order of the nodes, we show that any single source acyclic orientation $\overrightarrow{G}$ can be expressed as a sequence of direction assignments $\overrightarrow{Z}_i$ for $v_i$, for increasing values of $i$. Consider the following process: for decreasing values of $j$ remove $v_j$ from $\overrightarrow{G}$, and set $\overrightarrow{Z}_j$ equal to the current outgoing arcs

from $u$ in $\overrightarrow{G}$. The sequence of sets $\overrightarrow{Z}_1, \ldots, \overrightarrow{Z}_n$ will lead the algorithm to the discovery of $\overrightarrow{G}$. Note that each direction assignment $\overrightarrow{Z}_j$ is valid otherwise we have a cycle or a source different from $s$ in $\overrightarrow{G}$.

2. Each solution is acyclic, since each time we add a node $v_i$ and a valid direction assignment we do not introduce a cycle by definition of valid direction assignment. We have to show that $s$ is the unique source; any $v_j$ full in $\overrightarrow{G}_i$, with $j \leq i < n$, is not a source in $\overrightarrow{G}_i$: hence it is not a source in $\overrightarrow{G}_n = \overrightarrow{G}$. Indeed, for the good ordering definition, each node $v_j$ not full in $\overrightarrow{G}_i$, with $j \leq i$, has a neighbor in $V \setminus V_{\leq i}$: if it is a source in $\overrightarrow{G}_i$, when considering its last neighbor $v_z$ in the good order, it will not be a source anymore in $\overrightarrow{G}_z$.
3. Given any two solutions, looking at the primary recursion tree induced by Algorithm 1, they differ at least for the valid direction assignments branching in their least common ancestor.

□

**Theorem 1.** SSAO *can be solved with delay* $O(n \cdot m)$ *and space* $O(n^2)$.

*Proof.* We exploit the properties of the primary recursion tree induced by Algorithm 1. First of all, notice that each internal node has at least one child because of Property 1. This means that all the leaves correspond to a solution. Moreover, observe that all the leaves are at the same depth $n$, which is the height of the recursion tree. The first solution is clearly returned in time $O(n \cdot m)$, which is the height times the cost to get the first valid direction assignment for $v_i$. This is bounded by $O(m)$ time by applying Lemma 1. For any two consecutive solutions, the delay is bounded by the sum of the costs along a leaf-to-root path and the root-to-next-leaf path. The former is bounded by $O(n \cdot m)$: indeed the height of the tree is $O(n)$ and each time we 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 1.

The space is bounded by $O(n^2)$, that is the space occupancy of a root-to-leaf path in the primary recursion. Indeed, in this path we have to maintain $O(n)$ times the status of the `Generate` iterator, whose total space is bounded by $O(n)$.

□

## 4 Acyclic orientations (AO)

This section deals with the problem of enumerating all the acyclic orientations of an undirected graph $G$. The general scheme remains the same as discussed in Section 3. Differently from before, we have no restriction about the possible sources when adding $v_i$. Hence we redefine the concept of valid direction assignment as follows.

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

Referring to Figure 2, when adding $v_7$ to $\overrightarrow{G}_6$, this means that also the direction assignment $(v_4, v_7), (v_5, v_7), (v_6, v_7)$ is valid.

**Algorithm 3:** `acyclic`

**Input**: Graph $G(V, E)$, partial acyclic orientation $\overrightarrow{G}_{i-1}(V_{\leq i-1}, \overrightarrow{E})$, integer $i$
**Output**: Acyclic orientations of $G$ containing $\overrightarrow{G}_{i-1}(V_{\leq i-1}, \overrightarrow{E})$

**if** $i > n$ **then** output $\overrightarrow{G}_n$; **return** ;
Execute Algorithm 4;
**for** any valid direction assignment $\overrightarrow{Z}$ for $v_i$ starting from $\overrightarrow{Z} = X_i$ to $\overrightarrow{Z} = Y_i$ **do**
  `acyclic`$(G, \overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{Z}), i+1)$;

Moreover, 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 2.* For any $v_i$ there are always at least two valid direction assignments, i.e. $X_i = \{(x, v_i) : \ x \in N_<(v_i)\}$ and $Y_i = \{(v_i, x) : \ x \in N_<(v_i)\}$.

*Proof.* While adding $v_i$ to $\overrightarrow{G}_{i-1}$, adding the arcs of $X_i$ to $\overrightarrow{G}_{i-1}$ or adding the arcs of $Y_i$ does not create a cycle. Indeed, inductively the following facts hold. $\overrightarrow{G}_0$ does not contains a cycle. Assuming that $\overrightarrow{G}_{i-1}$ does not contain a cycle, any cycle should involve $v_i$. Since in the two orientations above $v_i$ is source or target, adopting one of these direction assignments cannot make $\overrightarrow{G}_i$ cyclic. □

Property 2 simply states that in Figure 2, both $\{(v_4, v_7), (v_5, v_7), (v_6, v_7)\}$ and $\{(v_7, v_4), (v_7, v_5), (v_7, v_6)\}$ are valid direction assignments when adding $v_7$ to $\overrightarrow{G}_6$. This is because $\overrightarrow{G}_6$ is acyclic and both of the new direction assignments cannot create cycles in the corresponding $\overrightarrow{G}_7$.

The actual scheme is summarized in Algorithm 3, whose starting call is `acyclic`$(G, \overrightarrow{G}_0, 1)$. At step $i$, given the acyclic orientation $\overrightarrow{G}_{i-1}(V_{\leq i-1}, \overrightarrow{E})$, each recursive call is of the kind `acyclic`$(G, \overrightarrow{G}_i, i+1)$ where $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{Z})$ is obtained by adding a valid direction assignment $\overrightarrow{Z}$. By Property 2, $\overrightarrow{Z} = X_i$ and $\overrightarrow{Z} = Y_i$ are always taken. The corresponding two recursive calls are done respectively at the beginning and at the end of the procedure. All the other possible valid direction assignments (if any) are explored in the other calls of the `for` cycle. When $i = n + 1$, all the nodes have been added and all the edges have been assigned a direction. In this case $\overrightarrow{G}_n$ is an acyclic orientation to be output.

**Lemma 3.** *Referring to Algorithm 3, 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. □

We introduce a method that, at running time, allows us to iterate just on valid direction assignments: in particular, this method gets the first valid direction assignment $X_i$ in $O(|N_<(v_i)|)$ time and the remaining ones with delay $O(m)$, one

**Algorithm 4:** Returning valid direction assignments

**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 $\overrightarrow{Z}$

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` $(G, \overrightarrow{G}_{i-1}, v_i, \overrightarrow{W} \cup \{(v_i, x_j)\}, j+1, R, B)$;

after the other. The point is that we spend $O(m)$ time and get a new solution. Here it is crucial that $Y_i$ is the last valid direction assignment for this purpose as it indicates when stopping the search for valid direction assignments. The motivation for this choice is given by the following lemma.

**Lemma 4.** *When iterating over valid direction assignments $\overrightarrow{Z}$, suppose that the first assignment $X_i$ is returned in $O(|N_<(v_i)|)$ time and the delay between any two consecutive valid direction assignments is $O(m)$. Then Algorithm 3 has delay $O(m)$.*

*Proof.* In the primary recursion tree, the internal nodes have at least two children, the leaves are all the solutions, and their depth is $n$. The solution in the first leaf is obtained by using always $X_i$ sets, each one corresponding to a node $v_i$, and the cost is $\sum_{i=1}^{n} O(|N_<(v_i)|) = O(m)$, using the hypothesis of the lemma.

The delay between two consecutive solutions is upper bounded by the sum of the costs of the recursive calls to go from a leaf to the next leaf in preorder. Now let $S$ and $T$ be the solutions in two consecutive leaves. The paths from the root to $S$ and to $T$ share the prefix until a recursive call $R_j$, corresponding to the recursion while adding $v_j$ to $\overrightarrow{G}_{j-1}$, for some $1 \leq j \leq n-1$. Let $S'$ and $T'$ be $R_j$'s children that are respectively the ancestors of $S$ and $T$. Note that the path from $S'$ to $S$ is made of $Y_i$ branches while the path from $T'$ to $T$ is made of $X_i$ branches ($j+1 \leq i \leq n$). The cost from $S$ to $S'$ is $O(n-j)$ as we do not need to check for further valid direction assignments after each $Y_i$. The cost from $S'$ to $T'$ is $O(m)$ by hypothesis as they are two consecutive valid direction assignments for $v_j$. The cost from $T'$ to $T$ is $O(\sum_{i=j+1}^{n} |N_<(v_i)|) = O(m)$ by hypothesis on the costs to get each $X_i$. □

In the next section we will provide a way of iterating over valid direction assignments fitting the hypothesis of Lemma 4.

### 4.1 Iterating over valid direction assignments

Given the node $v_i$ and the current acyclic directed graph $\overrightarrow{G}_{i-1}$, this section describes how to get all the valid direction assignments (i.e. such that adding one of them and $v_i$ to $\overrightarrow{G}_{i-1}$, we obtain a $\overrightarrow{G}_i$ which is still acyclic).

Algorithm 4 extends the current partial valid direction assignment $\overrightarrow{W}$ for $v_i$: for each edge in $\{v_i, x\}$ such that $x \in N_<(v_i)$, it adds the arc $(v_i, x)$ or the arc $(x, v_i)$ to $\overrightarrow{W}$ whether the partial direction assignment is still valid, i.e. no cycles are created. It explores all of these extensions.

More formally, given $\overrightarrow{G}_{i-1}(V_{\leq i}, \overrightarrow{E})$ and $v_i$, we consider the nodes $x_1, \ldots, x_k$ in $N_<(v_i)$ (where $|N_<(v_i)| = k$) one after the other. Initially, let $\overrightarrow{W}$ be an empty set. For increasing values of $j$, with $1 \leq j \leq k$, we do the following. If the arc $e \in \{(v_i, x_j), (x_j, v_i)\}$ does not create a cycle in $\overrightarrow{G}_{i-1}(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{W})$, this arc can be added to the ongoing solution $\overrightarrow{W}$, exploring recursively the case $\overrightarrow{W} = \overrightarrow{W} \cup \{e\}$.

Let $R$ and $B$ be respectively the set of nodes reachable from $v_i$ in $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{W})$ and the set of nodes leading to $v_i$ in $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{W})$. Adding the arc $(v_i, x_j)$ to $\overrightarrow{W}$ creates a cycle if and only if $x_j \notin B$. Analogously, adding the arc $(x_j, v_i)$ to $\overrightarrow{W}$ creates a cycle if and only if $x_j \notin R$. The scheme of the iterator is summarized by Algorithm 4. Notice that the first valid direction assignment produced is $X_i$ and the last valid direction assignment is $Y_i$, as required by Algorithm 3. 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 means respectively that the outdegree and the indegree of $v_i$ in $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{W})$ is zero.

We remark that there are no dead ends. Indeed, if $x_j \in R$ then $x_j \notin B$, meaning that even if the first call is skipped the second one is performed. Similarly $x_j \in B$ implies $x_j \notin R$. This means that each call produces at least another call unless the direction assignment is completed, that is, each call returns at least one solution.

**Lemma 5.** *Algorithm 4 returns the first valid direction assignment in time $O(|N_<(v_i)|)$, and the remaining ones with delay $O(m)$.*

*Proof.* Recall that the direction assignment $X_i$ is always returned first while $Y_i$ is returned last. For increasing values of $i$, adding the arcs of $X_i$ (respectively $Y_i$) does not create cycles: in this case no check is needed. In particular $X_i$ is returned in time $O(|N_<(v_i)|)$ since the update of $R$ is not needed and never performed in Algorithm 4 (because $\overrightarrow{W} \cap Y_i = \emptyset$, i.e. the outdegree of $v_i$ in $\overrightarrow{G}_i(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{W})$ is zero). Checking $\overrightarrow{W} \cap Y_i \neq \emptyset$ and $\overrightarrow{W} \cap X_i \neq \emptyset$ can be done in constant time: these are the out- and in-degree of $x_j$ in $\overrightarrow{G}(V_{\leq i}, \overrightarrow{E} \cup \overrightarrow{W})$ that can be updated while updating $\overrightarrow{W}$. The time is hence dominated by the cost to update $R$ and $B$. As in Lemma 1, this can be done by growing $R$ and $B$ continuing the same forward and backward traversals from $v_i$: since each arc is traversed at most once, the overall time to update $R$ and $B$ is $O(m)$. □

By combining Lemma 4 and Lemma 5, we obtain the following result.

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

## 5 Reducing to Single Source Acyclic Orientations

We show that the problems mentioned in Section 1 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 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.

As for AO, it can be obtained from weak MSAO by setting $S = V$. A direct reduction to SSAO is described in [14], although the paper does not provide an algorithm for SSAO.

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

**Theorem 3.** *Problems* SSAO*, weak* SSAO*, strong* MSAO*, and weak* MSAO *can be solved with delay* $O(m \cdot n)$ *and space* $O(n^2)$.

## 6 Conclusions

In this paper we have shown the first enumeration algorithms with guaranteed bounds for SSAO, weak SSAO, strong MSAO, and weak MSAO, whose delay is $O(m \cdot n)$ time and $O(n^2)$ space. The delay reduces to $O(m)$ in the case of AO, improving prior work. It would be interesting to reduce the delay of the former problems to $O(m)$ as well.

## References


1. N. Alon and M. Tarsi. Colorings and orientations of graphs. *Combinatorica*, 12(2):125–134, 1992.
2. N. Alon and Z. Tuza. The acyclic orientation game on random graphs. *Random Structures & Algorithms*, 6(2-3):261–268, 1995.
3. V. C. Barbosa and J. L. Szwarcfiter. Generating all the acyclic orientations of an undirected graph. *Information Processing Letters*, 72(1):71 – 74, 1999.
4. B. Benson, D. Chakrabarty, and P. Tetali. G-parking functions, acyclic orientations and spanning trees. *Discrete Mathematics*, 310(8):1340–1353, 2010.
5. A. Conte, R. Grossi, A. Marino, and R. Rizzi. Enumerating cyclic orientations of a graph. *IWOCA, 26th International Workshop on Combinatorial Algorithms*, 2015. To appear.
6. P. Erdős, G. Katona, and B. J. M. Társulat. *Theory of Graphs: Proceedings of the Colloquium Held at Tihany, Hungary, September 1966*. Academic Press, 1968.
7. B. Iriarte. Graph orientations and linear extensions. *DMTCS Proceedings*, pages 945–956, 2014.
8. D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, 27(3):119–123, 1988.
9. R. Johnson. Network reliability and acyclic orientations. *Networks*, 14(4):489–505, 1984.
10. N. Linial. Hard enumeration problems in geometry and combinatorics. *SIAM Journal on Algebraic Discrete Methods*, 7(2):331–335, 1986.
11. O. Pikhurko. Finding an unknown acyclic orientation of a given graph. *Combinatorics, Probability and Computing*, 19:121–131, 1 2010.
12. B. Roy. Nombre chromatique et plus longs chemins d'un graphe. *Revue française d'informatique et de recherche opérationnelle*, 1(5):129–132, 1967.
13. B. Schwikowski and E. Speckenmeyer. On enumerating all minimal solutions of feedback problems. *Discrete Applied Mathematics*, 117(1):253–265, 2002.
14. M. B. Squire. Generating the acyclic orientations of a graph. *Journal of Algorithms*, 26(2):275 – 290, 1998.
15. R. Stanley. Acyclic orientations of graphs. In I. Gessel and G.-C. Rota, editors, *Classic Papers in Combinatorics*, Modern Birkhäuser Classics, pages 453–460. Birkhäuser Boston, 1987.
16. R. P. Stanley. *What Is Enumerative Combinatorics?* Springer, 1986.
17. L. Vitaver. Determination of minimal coloring of vertices of a graph by means of boolean powers of the incidence matrix. In *Dokl. Akad. Nauk SSSR*, volume 147, page 728, 1962.