Enumerating Cyclic Orientations of a Graph

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

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

Abstract. Acyclic and cyclic orientations of an undirected graph have been widely studied for their importance: an orientation is acyclic if it assigns a direction to each edge so as to obtain a directed acyclic graph (DAG) with the same vertex set; it is cyclic otherwise. As far as we know, only the enumeration of acyclic orientations has been addressed in the literature. In this paper, we pose the problem of efficiently enumerating all the *cyclic* orientations of an undirected connected graph with n vertices and m edges, observing that it cannot be solved using algorithmic techniques previously employed for enumerating acyclic orientations. We show that the problem is of independent interest from both combinatorial and algorithmic points of view, and that each cyclic orientation can be listed with $\tilde{O}(m)$ delay time. Space usage is $O(m)$ with an additional setup cost of $O(n^2)$ time before the enumeration begins, or $O(mn)$ with a setup cost of $\tilde{O}(m)$ time.

1 Introduction

Given an undirected graph $G(V, E)$ with $n = |V|$ vertices and $m = |E|$ edges, an orientation transforms G into a directed graph $\overrightarrow{G}$ by assigning a direction to each edge. That is, an orientation of G is the directed graph $\overrightarrow{G}(V, \overrightarrow{E})$ such that the vertex set V is the same as G, and the edge set $\overrightarrow{E}$ is an orientation of E: exactly one direction between $(u, v) \in \overrightarrow{E}$ and $(v, u) \in \overrightarrow{E}$ holds for any undirected edge $\{u, v\} \in E$. An orientation $\overrightarrow{G}$ is *acyclic* when $\overrightarrow{G}$ does not contain any directed cycles, so $\overrightarrow{G}$ is a directed acyclic graph (DAG); otherwise we say that the orientation $\overrightarrow{G}$ is cyclic.

Acyclic orientations of undirected graphs have been studied in depth. Many results concern the number of such orientations: Stanley [14] shows how, given a graph G and its chromatic number χ, the number of acyclic orientations of G can be computed by using the chromatic polynomial (a special case of Tutte's polynomial). Other results rely on the so called acyclic orientation game: Alon et al. [1] inquire about the number of edges that have to be examined in order to identify an acyclic orientation of a random graph G; Pikhurko [10] gives an upper bound on this number of edges in general graphs. The counting problem is known to be #P-complete [8] and enumeration algorithms that list all the acyclic orientations of a graph are given in [2] and [13].

We consider *cyclic* orientations, which have been also studied from many points of view. Counting them is co-#P-complete [8]. In Fisher et al. [5], given

a graph G and an acyclic orientation of it, the number of *dependent edges*, i.e. edges generating a cycle if reversed, has been studied. This number of edges implicitly gives a hint on the number of cyclic orientations in a graph. In [11] an algorithm has been given to make inference about causal structure in cyclic graphs. In [12] directed cyclic graphs are used to model economic processes, and an algorithm is given to characterize conditional independence constraints of these processes.

In this paper we address the problem of enumerating all the cyclic orientations of a graph.

Problem 1. Enumerating the set of all the directed graphs $\overrightarrow{G}$ that are cyclic orientations of an undirected graph G.

We analyze the cost of an enumeration algorithm for Problem 1 in terms of its *setup* cost, meant as the initialization time before the algorithm is able to lists the solutions, and its *delay* cost, which is the worst-case time between any two consecutively enumerated solutions (e.g. [7]). We are interested in algorithms with guaranteed $\tilde{O}(m)$ delay, where the $\tilde{O}$ notation ignores polylogs.

A naive solution to Problem 1 uses the fact that enumeration algorithms exist for listing acyclic orientations [2,13]. It enumerates the cyclic orientations by difference, namely, by enumerating all the 2^m possible edge orientations and removing the α acyclic ones. However, this solution does not guarantee any polynomial delay, as the number $\beta = 2^m - \alpha$ if cyclic orientations can be much larger or much smaller than the number α of acyclic ones. For example, a tree with m edges has $\alpha = 2^m$ and $\beta = 0$. On the other extreme of the situation, we have cliques. An oriented clique is also called a *tournament*, and a *transitive tournament* is a tournament with no cycles. A clique of n nodes (and $m = \frac{n \cdot (n-1)}{2}$ edges) can generate 2^m different tournaments, out of which exactly $\alpha = n!$ will be transitive tournaments [9]. As 2^m grows faster than $n!$, we have that the ratio α/β tends to 0 for increasing n, where $\beta = 2^m - \alpha$.

To the best of our knowledge, an enumeration algorithm for Problem 1 with guaranteed $\tilde{O}(m)$ delay is still missing. We provide such an algorithm in this paper, namely, listing each cyclic orientation with $\tilde{O}(m)$ delay time. Space usage is $O(m)$ memory cells with a setup cost of $O(n^2)$ time, or $O(mn)$ memory cells with a setup cost of $\tilde{O}(m)$ time. Interestingly, Problem 1 reveals to be a rich source for enumerations techniques, and our solution offers new combinatorial and algorithmic techniques when compared to previous work on the enumeration of acyclic orientations [2,13].

In the following, for the sake of clarity, we will call *edges* the elements of E (undirected graph) and *arcs* the elements of $\overrightarrow{E}$ (directed graph). We will assume that the graph in input G is connected and we will denote as $n = |V|$ and $m = |E|$ respectively its number of nodes and edges.

The paper is organized as follows. Section 2 gives an overview of our enumeration algorithm. Section 3 describes the initialization steps, and shows how to reduce the problem from the input graph to a suitable multi-graph that guarantees to have a chordless cycle (hole) of logarithmic size. The latter is crucial to obtain the claimed delay. Section 4 shows how to enumerate in the multigraph

and obtain the cyclic orientations for the input graph. Section 5 describes how to absorb the setup cost using more space. Finally, some conclusions are drawn in Section 6.

2 Algorithm overview

The intuition behind our algorithm for an undirected connected graph $G(V_G, E_C)$ is the following one. Suppose that G is cyclic, otherwise there are no cyclic orientations. Consider one cycle[3] $C(V_C, E_C)$ in G: we can orient its edges in two ways so that the resulting $\overrightarrow{C}$ is a directed cycle. At this point, *any* orientation of the remaining edges, e.g. those in $E_G \setminus E_C$, will give a cyclic orientation of G. Thus, the interesting cases are when the resulting $\overrightarrow{C}$ is not a directed cycle.

The idea is first to 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 could have a cyclic orientation even if $\overrightarrow{C}$ is *not* a directed cycle.

Therefore we must consider some cases. One easy case is that the orientation of $E_G \setminus E_C$ already produces a directed cycle: any orientation of E_C will give a cyclic orientation of G. Another easy case, as seen above, is for the two orientations of E_C such that $\overrightarrow{C}$ is a directed cycle: any orientation of $E_G \setminus E_C$ will give a cyclic orientation of G. It remains the case when the orientations of both $E_G \setminus E_C$ and E_C are individually acyclic: when put together, we might have or not a directed cycle in the resulting orientation of G. To deal with the latter case, we need to "massage" G and transform it into a multigraph as follows. We refer the reader to Algorithm 1.

Algorithm setup is performed as described in Section 3. We preprocess G with dead-ends removal and edge chain compression. The result is an undirected connected multigraph $M(V_M, E_M)$, where the edges are labeled as *simple* and *chain*. After that we find a chordless cycle of logarithmic size C in M, and remove E_C from E_M, obtaining the labeled multigraph $M'(V_M, E'_M)$, where $E'_M = E_M \setminus E_C$.

Enumerating cyclic orientations described in Section 4 exploits the property (which we will show later) that finding cyclic orientations of G corresponds to finding particular orientations in M', called *extended cyclic orientations*, and of C, called *legal orientations*. In the **for** loop, these orientations of M' and C are enumerated so as to find all the cyclic orientations of G. As we will see for the latter task, it is important to have C of logarithmic size to guarantee our claimed delay.

[3] This will actually be a chordless cycle of logarithmic size (called log-hole).

Algorithm 1: 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 3):
Remove dead-ends (nodes of degree 1) recursively from G
$M(V_M, E_M) \leftarrow$ replace G's maximal paths of degree-2 nodes with chain edges
$C(V_C, E_C) \leftarrow$ a log-hole of M
$M'(V_M, E'_M) \leftarrow$ delete the edges of C from M, i.e. $E'_M = E_M \setminus E_C$

Enumerate cyclic orientations (Section 4):
for each extended orientation $\overrightarrow{M'}$ of multigraph M' **do**
  for each legal orientation $\overrightarrow{C}$ of log-hole C (see Algorithm 2) **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 $\overrightarrow{M''}$

3 Algorithm Setup

3.1 Reducing the problem to extended cyclic orientations

In the following we show how to reduce Problem 1 to an extended version that allows us to neglect dead-ends and chains.

Dead-end removal. Given an undirected graph $G(V, E)$, a dead-end is a node of degree 1. We recursively remove all *dead-ends*, so that all the surviving nodes have degree 2 or greater. By removing these nodes and computing the cyclic orientations in the cleaned graph, we can still generate solutions for the original graph by using both orientations of the unique incident edge to each dead-end, as emphasized by the following lemma, whose proof is straightforward.

Lemma 1. *Let G be a graph, u a dead-end and $\{u, v\}$ its unique incident edge. Let G' be the graph G without u and the edge $\{u, v\}$. The set of all the cyclic orientations of G is composed by the orientations $\overrightarrow{G}'(V' \cup \{u\}, \overrightarrow{E}' \cup \{(u, v)\})$ and $\overrightarrow{G}'(V' \cup \{u\}, \overrightarrow{E}' \cup \{(v, u)\})$, for all the cyclic orientations $\overrightarrow{G}'(V', \overrightarrow{E}')$ of G'.*

The dead-end removal can be done by performing a DFS recursive traversal of G, starting from an arbitrary node x. Every time a node of degree 1 is visited, it is removed from the graph. When the recursion ends in a node, the latter is removed if all of its neighbors have been removed except one (which is its parent in the DFS tree). Finally, when the traversal ends, it might be that the node from which we started has degree 1. To complete the process, if x has now degree 1, we remove it from the graph. The DFS of G and the removal of its dead-ends can be done in $O(m)$.

The rationale for removing dead-ends is to have shorter cycles: for example, consider a "necklace" graph with n nodes, for n even, such that $n/2$ nodes form a cycle, and the remaining $n/2$ nodes have degree 1 and are attached to one of the

nodes in the cycle, such that each node in the cycle has degree 3 and is connected to one node of degree 1. With the removal of dead-ends, the cycle has only nodes of degree 2 and can be compressed as discussed in the next paragraph.

Chain compression. It consists in finding all the maximal paths $v_1, \ldots, v_k$ where v_i has degree 2 (with $2 \leq i \leq k-1$), and replacing each of them, called *chain*, with just one edge, called *chain edge*. It is easy to see that this task can be accomplished in $O(m)$ time by traversing the graph G in a DFS fashion from a node of degree ≥ 3. The output is an undirected connected multigraph $M(V_M, E_M)$, where $V_M \subseteq V$ are the nodes of V whose degree is ≥ 3, and E_M are the chain edges plus all the edges in E which are not part of a chain.[4] The latter ones are called simple edges to distinguish them from the chain edges. In the rest of the paper, M will be seen as a multigraph where $|V_M| \geq 4$ and each of the edges has a label in {simple, chain}, since it might contain parallel edges or loops. For this, we define the concept of *extended orientation* as follows.

Definition 1 (Extended Orientation). For a multigraph $M(V_M, E_M)$ having self loops and edges labeled as simple and chain, an *extended orientation* $\overrightarrow{M}(V_M, \overrightarrow{E_M})$ is an orientation $\overrightarrow{E_M}$ of its edge set E_M: for any simple edge $\{u, v\}$, exactly one direction between (u, v) and (v, u) holds; for any chain edge $\{u, v\}$, either is *broken*, or exactly one direction between (u, v) and (v, u) holds. A directed cycle in $\overrightarrow{M}$ cannot contain a broken edge.

Broken edges correspond to chain edges that, when expanded as edges of G, do not have an orientation as a directed path. This means that they cannot be traversed in either direction. Note that this situation cannot happen for simple edges. The following lemma holds.

Lemma 2. *If we have an algorithm that list all the extended cyclic orientations of $M(V_M, E_M)$ with delay $f(|E_M|)$, for some $f : \mathbb{R} \to \mathbb{R}$, then we have an algorithm that lists all the acyclic 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 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 has no direction assigned, i.e. *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 acyclic orientations of the graph $G(V, E)$ with delay $O(f(|E_M|) + |E|)$. □

[4] The degenerate case of M with ≤ 3 nodes can be handled separately.

Lemma 2 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.

3.2 Logarithmically bounded hole

Given the labeled multigraph obtained in Section 3.1, namely $M(V_M, E_M)$, we perform the following two steps.

1. **Finding a log-hole.** Find a *logarithmically bounded hole* (hereafter, log-hole) $C(V_C, E_C)$ in $M(V_M, E_M)$: it 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.[5]
2. **Removing the log-hole.** Remove the edges in E_C from M, obtaining $M'(V_M, E'_M)$, where $E'_M = E_M - E_C$.

Properties of the log-hole. Since M is a multigraph with self-loops, a log-hole $C(V_C, E_C)$ of M can potentially be a self-loop. In any case, the following well-known result holds.

Lemma 3 (Logarithmic girth [3,4]). *Let $G(V, E)$ be a graph in which every node has degree at least 3. The girth of G is $\leq 2\lceil \log |V| \rceil$.*

Lemma 3 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 finding the girth, that is performing a BFS on each node of the multigraph M to identify the shortest cycle that contains that node, in time $O(|V_M| \cdot |E_M|)$. By applying the algorithm in [6], which easily extends to multigraphs, we compute C in time $O(|V_M|^2)$: in this case, if chords are present in the found C, in time $O(\log |V_M|)$ we can check whether C includes a smaller cycle and redefine C accordingly.

4 Enumerating cyclic orientations

We now want to list all the cyclic orientations of the input graph G. By Lemma 2 this is equivalent to listing the extended cyclic orientations of the corresponding labeled multigraph $M(V_M, E_M)$ obtained from G by dead-end removal and chain compression. 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 an algorithm that is organized as follows.

[5] Minimum cycle means the cycle having minimum number of edges (i.e. a self loop), where chain edges count just one like normal edges.

1. **Finding extended orientations.** Enumerate all extended orientations (not necessarily cyclic) $\overrightarrow{M'}$ of the multigraph M'.
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.

Finding extended orientations. This is now 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 b broken edges in M', where $s + m = |E'_M|$, this generates all possible $2^s 3^b$ extended orientations. Each of them can be easily listed in $O(|E'_M|)$ delay (actually less, but this is not the dominant cost).

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 2. *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 counterclockwise. Moreover, adding just an oriented subset of edges $D \subseteq C$ to $\overrightarrow{M'}$ can 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 2. 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 2, 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 the starting $\overrightarrow{M}$, 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 **??** we have $|V_C| < 2\lceil \log |V_M| \rceil + 1$, 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. Namely, for each edge $\{u, v\}$ in E_C, if $\{u, v\}$ is simple we try the two possible direction assignments, while if it is a chain we also try the broken assignment. In order to avoid dead-end recursive calls, after each assignment we update the reachability matrix R and perform the recursive call only if this partial direction assignment will produce at least a solution: both the update of R and the dead-end check can be done in $O(\log^2 |V_M|)$ (that is, the size of R).

Scheme for legal orientations. The steps are shown by Algorithm 2. At the beginning the reachability matrix R is computed and is 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: 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 $\overrightarrow{M}'$.

The reachability and cyclicity checks are done by updating and checking the reachability matrix R (and restoring R when needed). 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 amount of time by looking for a pair of nodes in $\{c_1, \ldots c_j\}$ x',y' such $R(x', y') = R(y', y') = 1$.

Lemma 4. *Algorithm 2 outputs in $O(|E'_M| \log |V_M|)$ time the first legal orientation of C, and each of the remaining ones with $O(\log^3 |V_M|)$ delay.*

Proof. Before calling the `LegalOrientations` procedure we have to compute the reachability matrix from scratch and this costs $O(|E'_M| \log |V_M|)$ time. In the following we will bound the delay between two outputs returned by the `LegalOrientations` procedure. Firstly, note that each call produces at least one solution. This is true when $j = 1$ since we have two possible legal orientations

Algorithm 2: Returning all legal orientations of C

Input: $\overrightarrow{M'}(V_M, \overrightarrow{E'}_M)$ acyclic, a cycle $C(V_C, E_C)$ 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**
  - $u \leftarrow c_j, \quad v \leftarrow c_{j+1}$
  - $R_1 \leftarrow R$ 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` $(\overrightarrow{C}', j + 1, R, I \cup \{\{u, v\}\})$

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 means that in the recursion tree, every internal node has at least a 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 $O(\log^2 |V_M|)$, we can conclude that the 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 5 (Correctness).

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

Proof. 1. Any extended cyclic orientation $\overrightarrow{M}$ can be seen as the union $\overrightarrow{M}''$ of $\overrightarrow{M'}$ and $\overrightarrow{C}$, which are two edge disjoint directed subgraphs. Our algorithm enumerates all the extended orientations of M', and, for each of them, all legal extended orientations $\overrightarrow{C}$: if there is a cycle in $\overrightarrow{M}$ involving only edges in E'_M all the extended orientations of C are legal; otherwise just the extended

orientations $\overrightarrow{C}$ of C whose arcs create a cycle in $\overrightarrow{M}''$ are legal. Hence any extended cyclic orientation $\overrightarrow{M}$ is output.
2. Any output solution is an extended orientation: each edge in M' and in C has exactly one direction or is broken. Moreover, any output solution contains at least a cycle: if there is not a cycle in M', a cycle is created involving the edges in C.
3. All the extended orientations $\overrightarrow{M} = \overrightarrow{M}''$ in output differ for at least an arc in $\overrightarrow{E'}_M$ or an arc in $\overrightarrow{E_C}$. Hence there are no duplicate solutions.

□

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

Lemma 6. *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 2. By Lemma 4 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 all the phases: in particular in Algorithm 2 the space is $O(\log^2 |V_M|)$, because of the reachability matrix R, which is smaller than $O(|E_M|)$. □

Applying **??** and **??**, and considering the setup cost in Section 3 ($|V_M| \leq |V|$ and $|E_M| \leq |E|$), we can conclude as follows.

Theorem 1. *Algorithm 1 lists all cyclic orientations of $G(V, E)$ with setup cost $O(|V|^2)$, and delay $\tilde{O}(|E|)$. The space usage is $O(|E|)$ memory cells.*

5 Absorbing the setup cost

In this section, we show how to modify our approach to get a setup time equal to the delay, requiring space $\Theta(|V| \cdot |E|)$.

Theorem 2. *All cyclic orientations of $G(V, E)$ can be listed with setup cost $\tilde{O}(|E|)$, delay $\tilde{O}(|E|)$, and space usage of $\Theta(|V| \cdot |E|)$ memory cells.*

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. Note that C_u is a log-hole as required. Now, if $|C_u| < \log n$, 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 1, with a setup cost of $O(mn)$ and $\tilde{O}(m)$ delay (i.e. Algorithm 1). 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, $O(mn)$ memory cells to store up to n solutions.

We also have an amortized cost using the lemma below, where $f(x) = \tilde{O}(x)$ and $s = |V|$.

Lemma 7. *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 [6] 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

$$\frac{O(2^s \cdot f(|E_M|) + s \cdot |V_M|)}{2^s} = O\left(f(|E_M|) + |E_M| \cdot \frac{s}{2^s}\right)$$

□

6 Conclusions

In this paper we considered the problem of efficiently enumerating cyclic orientations of graphs. The problem is interesting from a combinatorial and algorithmic point of view, as the fraction of cyclic orientations over all the possible orientations can be as small as 0 or very close to 1. We provided an efficient algorithm to enumerate the solutions with delay $\tilde{O}(m)$ and overall complexity $\tilde{O}(\alpha \cdot m)$, with α being the number of solutions.

References

1. N. Alon and Z. Tuza. The acyclic orientation game on random graphs. *Random Structures & Algorithms*, 6(2-3):261–268, 1995.
2. V. C. Barbosa and J. L. Szwarcfiter. Generating all the acyclic orientations of an undirected graph. *Information Processing Letters*, 72(1):71 – 74, 1999.
3. B. Bollobas. *Extremal Graph Theory*. Dover Publications, Incorporated, 2004.
4. P. Erdős and L. Pósa. On the maximal number of disjoint circuits of a graph. *Publ. Math. Debrecen*, 9:3–12, 1962.
5. D. C. Fisher, K. Fraughnaugh, L. Langley, and D. B. West. The number of dependent arcs in an acyclic orientation. *Journal of Combinatorial Theory, Series B*, 71(1):73 – 78, 1997.
6. A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978.
7. D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, 27(3):119–123, 1988.
8. N. Linial. Hard enumeration problems in geometry and combinatorics. *SIAM Journal on Algebraic Discrete Methods*, 7(2):331–335, 1986.
9. J. Moon. *Topics on tournaments*. Athena series: Selected topics in mathematics. Holt, Rinehart and Winston, 1968.
10. O. Pikhurko. Finding an unknown acyclic orientation of a given graph. *Combinatorics, Probability and Computing*, 19:121–131, 1 2010.
11. T. Richardson. A discovery algorithm for directed cyclic graphs. In *Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence*, UAI'96, pages 454–461, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.
12. P. Spirtes. Directed cyclic graphical representations of feedback models. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, UAI'95, pages 491–498, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.
13. M. B. Squire. Generating the acyclic orientations of a graph. *Journal of Algorithms*, 26(2):275 – 290, 1998.
14. 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.