On the Parameterized Complexity of Computing *st*-Orientations with Few Transitive Edges

Carla Binucci
Department of Engineering, University of Perugia, Italy

Giuseppe Liotta
Department of Engineering, University of Perugia, Italy

Fabrizio Montecchiani
Department of Engineering, University of Perugia, Italy

Giacomo Ortali
Department of Engineering, University of Perugia, Italy

Tommaso Piselli
Department of Engineering, University of Perugia, Italy

Abstract

Orienting the edges of an undirected graph such that the resulting digraph satisfies some given constraints is a classical problem in graph theory, with multiple algorithmic applications. In particular, an *st*-orientation orients each edge of the input graph such that the resulting digraph is acyclic, and it contains a single source s and a single sink t. Computing an *st*-orientation of a graph can be done efficiently, and it finds notable applications in graph algorithms and in particular in graph drawing. On the other hand, finding an *st*-orientation with at most k transitive edges is more challenging and it was recently proven to be NP-hard already when $k = 0$. We strengthen this result by showing that the problem remains NP-hard even for graphs of bounded diameter, and for graphs of bounded vertex degree. These computational lower bounds naturally raise the question about which structural parameters can lead to tractable parameterizations of the problem. Our main result is a fixed-parameter tractable algorithm parameterized by treewidth.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Mathematics of computing → Graph algorithms

Keywords and phrases *st*-orientations, parameterized complexity, graph drawing

Digital Object Identifier 10.4230/LIPIcs.MFCS.2023.18

Related Version *Full Version*: `http://arxiv.org/abs/2306.03196` [2]

Funding Research partially supported by (i) University of Perugia, Fondi di Ricerca di Ateneo, edizione 2021, project "AIDMIX - Artificial Intelligence for Decision making: Methods for Interpretability and eXplainability", (ii) Progetto RICBA22CB *"Modelli, algoritmi e sistemi per la visualizzazione e l'analisi di grafi e reti"*.

1 Introduction

An orientation of an undirected graph is an assignment of a direction to each edge, turning the initial graph into a directed graph (or *digraph* for short). Notable examples of orientations are acyclic orientations, which guarantee the resulting digraph to be acyclic; transitive orientations, which make the resulting digraph its own transitive closure; and Eulerian orientations, in which each vertex has equal in-degree and out-degree. Of particular interest for our research are certain constrained acyclic orientations, which find applications in several domains, including graph planarity and graph drawing. More specifically, given a graph $G = (V, E)$ and two vertices $s, t \in V$, an *st-orientation* of G, also known as *bipolar*

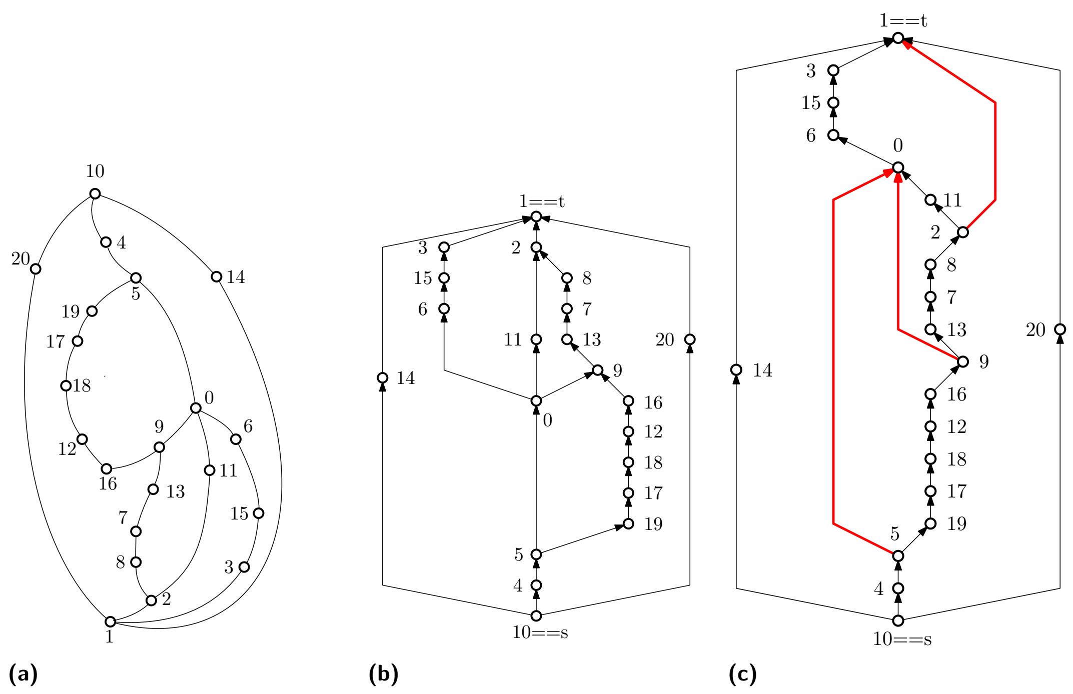


Figure 1 (a): An undirected graph G with randomly labeled vertices. (b)-(c): Two polyline drawings of G computed by using different st-orientations. The drawing in (b) uses an st-orientation without transitive edges and it has smaller area and number of bends than the drawing in (c).

orientation, is an orientation of its edges such that the corresponding digraph is acyclic and contains a single source s and a single sink t. It is well-known that G admits an st-orientation if and only if it is biconnected after the addition of the edge st (if not already present). The computation of an st-numbering (an equivalent concept defined on the vertices of the graph) is for instance part of the quadratic-time planarity testing algorithm by Lempel, Even and Cederbaum [11]. Later, Even and Tarjan [8] showed how to compute an st-numbering in linear time, and used this result to derive a linear-time planarity testing algorithm. In the field of graph drawing, bipolar orientations are a central algorithmic tool to compute different types of layouts, including visibility representations, polyline drawings, dominance drawings, and orthogonal drawings (see [5, 9] for references). On a similar note, a notable result states that a planar digraph admits an upward planar drawing if and only if it is the subgraph of a planar st-graph, that is, a planar digraph with a bipolar orientation [6].

Recently, Binucci, Didimo and Patrignani [1] focused on st-orientations with no transitive edges. We recall that an edge uv is *transitive* if the digraph contains a path directed from u to v; for example, the bold (red) edges in Figure 1c are transitive, see also Section 2 for formal definitions. Besides being of theoretical interest, such orientations, when they exist, can be used to compute readable and compact drawings of graphs [1]. For example, a classical graph drawing algorithm relies on st-orientations to compute polyline representations of planar graphs. The algorithm is such that both the height and the number of bends of the representations can be reduced by computing st-orientations with few transitive edges. See Algorithm `Polyline` in [5] for details and Figure 1 for an example.

Unfortunately, while an st-orientation of an n-vertex graph can be computed in $O(n)$ time, computing one that has the minimum number of transitive edges is much more challenging from a computational perspective. Namely, Binucci et al. [1] prove that the problem of deciding whether an st-orientation with no transitive edges exists is NP-complete, and provide an ILP model for planar graphs.

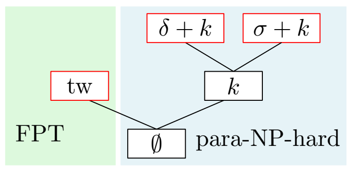


Figure 2 The complexity landscape of the ST-ORIENTATION problem. The symbols tw, δ, and σ denote the treewidth, the maximum vertex degree, and the diameter of the graph, respectively. The boxes with red boundaries denote the new results presented in this paper.

Contribution. We study the parameterized complexity of finding *st*-orientations with few transitive edges. More formally, given a graph G and an integer k, the ST-ORIENTATION problem asks for an *st*-orientation of G with at most k transitive edges (see also Section 2). As already discussed, ST-ORIENTATION is para-NP-hard by the natural parameter k [1]. We strengthen this result by showing that, for $k = 0$, ST-ORIENTATION remains NP-hard even for graphs of diameter at most six, and for graphs of vertex degree at most four. In light of these computational lower bounds, we seek for structural parameters that can lead to tractable parameterizations of the problem. Our main result is a fixed-parameter tractable algorithm for ST-ORIENTATION parameterized by treewidth, a central parameter in the parameterized complexity analysis (see [7, 12]). Figure 2 depicts a summary of the computational complexity results known for the ST-ORIENTATION problem.

It is worth remarking that by Courcelle's theorem one can derive an (implicit) FPT algorithm parameterized by treewidth and k, while we provide an explicit algorithm parameterized by treewidth only. The main challenge in applying dynamic programming over a tree decomposition is that the algorithm must know if adding an edge to the graph may cause previously forgotten non-transitive edges to become transitive, and, if so, how many of them. To tackle this difficulty, we describe an approach that avoids storing information about all edges that may potentially become transitive; instead, the algorithm guesses the edges that will be transitive in a candidate solution and ensures that no other edge will become transitive in the course of the algorithm. Our technique can be easily adapted to handle more general constraints on the sought orientation, for instance the presence of multiple sources and sinks.

Paper structure. We begin with preliminary definitions and basic tools, which can be found in Section 2. In Section 3 we describe our main result, an FPT algorithm for the ST-ORIENTATION problem parameterized by treewidth. Section 4 contains our second contribution, namely we adapt the NP-hardness proof in [1] to prove that the result holds also for graphs that have bounded diameter and for graphs with bounded vertex degree. In the latter case, the graphs used in the reduction not only have bounded vertex degree (at most four), but are also subdivisions of triconnected graphs. In Section 5 we list some interesting open problems that stem from our research.

For space reasons, some proofs have been omitted, and the corresponding statements are marked with ($\star$).

2 Preliminaries

Edge orientations. Let $G = (V, E)$ be an undirected graph. An *orientation* O of G is an assignment of a *direction*, also called *orientation*, to each edge of G. We denote by $D_O(G)$ the digraph obtained from G by applying the orientation O. For each undirected pair $(u, v) \in E$, we write uv if (u, v) is oriented from u to v in $D_O(G)$, and we write vu otherwise. A directed path from a vertex u to a vertex v is denoted by $u \rightsquigarrow v$. A vertex of $D_O(G)$ is a *source* (*sink*) if all its edges are outgoing (incoming). An edge uv of $D_O(G)$ is *transitive* if $D_O(G)$ contains a directed path $u \rightsquigarrow v$ distinct from the edge uv. A digraph $D_O(G)$ is an *st-graph* if: (i) it contains a single source s and a single sink t, and (ii) it is acyclic. An orientation O such that $D_O(G)$ is an st-graph is called an *st-orientation*.

ST-ORIENTATION
Input: An undirected graph $G = (V, E)$, two vertices $s, t \in V$, and an integer $k \geq 1$.
Output: An st-orientation O of G such that the resulting digraph $D_O(G)$ contains at most k transitive edges.

We recall that ST-ORIENTATION is NP-complete already for $k = 0$ [1], which hinders tractability in the parameter k. Also, in what follows, we always assume that the input graph G is connected, otherwise we can immediately reject the instance as any orientation would give rise to at least one source and one sink for each connected component of G.

Tree-decompositions. Let $(\mathcal{X}, T)$ be a pair such that $\mathcal{X} = \{X_i\}_{i\in[\ell]}$ is a collection of subsets of vertices of a graph $G = (V, E)$, called *bags*, and T is a tree whose nodes are in one-to-one correspondence with the elements of $\mathcal{X}$. When this creates no ambiguity, X_i will denote both a bag of $\mathcal{X}$ and the node of T whose corresponding bag is X_i. The pair $(\mathcal{X}, T)$ is a *tree-decomposition* of G if:

1. $\bigcup_{i\in[\ell]} X_i = V$,
2. For every edge uv of G, there exists a bag X_i that contains both u and v, and
3. For every vertex v of G, the set of nodes of T whose bags contain v induces a non-empty (connected) subtree of T.

The *width* of $(\mathcal{X}, T)$ is $\max_{i=1}^{\ell} |X_i| - 1$, while the *treewidth* of G, denoted by $\mathrm{tw}(G)$, is the minimum width over all tree-decompositions of G. The problem of computing a tree-decomposition of width $\mathrm{tw}(G)$ is fixed-parameter tractable in $\mathrm{tw}(G)$ [3]. A tree-decomposition $(\mathcal{X}, T)$ of a graph G is *nice* if T is a rooted binary tree with the following additional properties [4]:

1. If a node X_i of T has two children whose bags are X_j and $X_{j'}$, then $X_i = X_j = X_{j'}$. In this case, X_i is a *join bag*.
2. If a node X_i of T has only one child X_j, then $X_i \neq X_j$ and there exists a vertex $v \in G$ such that either $X_i = X_j \cup \{v\}$ or $X_i \cup \{v\} = X_j$. In the former case X_i is an *introduce bag*, while in the latter case X_i is a *forget bag*.
3. If a node X_i is the root or a leaf of T, then $X_i = \emptyset$. In this case, X_i is a *leaf bag*.

Given a tree-decomposition of width ω of G, a nice tree-decomposition of G with the same width can be computed in $O(\omega \cdot n)$ time [10].

3 The st-Orientation Problem Parameterized by Treewidth

In this section, we describe a fixed-parameter tractable algorithm for ST-ORIENTATION parameterized by treewidth. In fact, the algorithm we propose can solve a slightly more general problem. Namely, it does not assume that s and t are part of the input, but it looks for an st-orientation in which the source and the sink can be any pair of vertices of the input graph. However, if s and t are prescribed, a simple check can be added to the algorithm (we will highlight the crucial point in which the check is needed) to ensure this property.

Let $G = (V, E)$ be an undirected graph. A *solution* of the ST-ORIENTATION problem is an orientation O of G such that $D_O(G)$ is an st-graph with at most k transitive edges. Let $(\mathcal{X}, T)$ be a tree-decomposition of G of width ω. For a bag $X_i \in \mathcal{X}$, we denote by $G[X_i]$ the

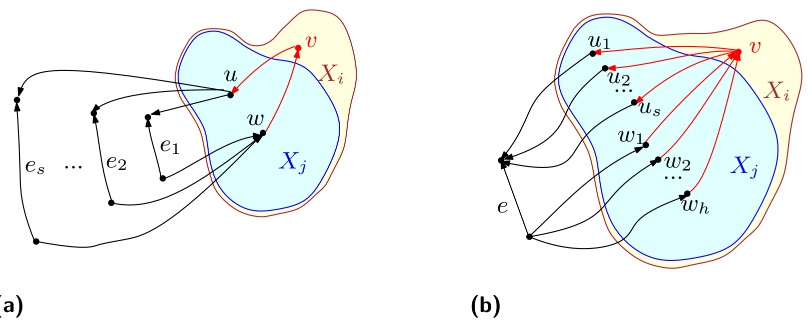


Figure 3 (a) The directed edges wv and vu make all edges $e_1, ..., e_s$ transitive. (b) Each pair of directed edges w_pv and v_qu, for $p \in [1, h]$ and $q \in [1, s]$, makes e transitive.

subgraph of G induced by the vertices of X_i, and by T_i the subtree of T rooted at X_i. Also, we denote by G_i the subgraph of G induced by all the vertices in the bags of T_i. We adopt a dynamic-programming approach performing a bottom-up traversal of T. The solution space is encoded into records associated with the bags of T, which we describe in the next section.

3.1 Encoding solutions

Before describing the records stored for each bag, we highlight the main challenges about how to encode the partial solutions computed throughout the course of the algorithm. Let v be a vertex introduced in a bag X_i. Adding v and its incident edges to a partial solution may either turn many (possibly linearly many) forgotten edges into transitive edges and/or it may make the same forgotten edge transitive with respect to arbitrarily many different paths. This is schematically illustrated in Figure 3, where X_i and its child bag X_j are highlighted by shaded regions. In Figure 3a, $e_1, \dots, e_s$ are forgotten edges, i.e., edges in G_i but not in $G[X_i]$; if we orient edge (u, v) from v to u and edge (v, w) from w to v all edges $e_1, \dots, e_s$ become transitive. In Figure 3b, e is a forgotten edge, while $u_1, \dots, u_s$ and $w_1, \dots, w_h$ are vertices of bag X_j; orienting the edges (w_p, v) from w_p to v ($1 \le p \le h$) and the edges (v, u_q) from v to u_q, turns e into a transitive edge with respect to $h \times s$ different paths. In case of Figure 3a the algorithm cannot afford reconsidering the forgotten edges as they can be arbitrarily many. In case of Figure 3b the algorithm should avoid counting e multiple times (for each newly created path). To overcome these issues, the algorithm guesses the edges that are transitive in a candidate solution and verifies that no other edge can become transitive during the bottom-up visit of T. This is done by suitable records, describe below.

Let O be a solution and consider a bag $X_i \in \mathcal{X}$. The *record* R_i of X_i that *encodes* O represents the interface of the solution O with respect to X_i. For ease of notation, the restriction of $D_O(G)$ to G_i is denoted by D_i, and similarly the restriction to $G[X_i]$ is $D[X_i]$. Record R_i stores the following information.

- $\mathcal{O}_i$ which is the orientation of $D[X_i]$.
- $\mathcal{A}_i$ which is the subset of the edges of $D[X_i]$ that are transitive in $D_O(G)$. We call such edges *admissible transitive edges* or simply *admissible edges*. The edges of G_i not in $\mathcal{A}_i$ are called *non-admissible*. We remark that an edge of $\mathcal{A}_i$ may not be transitive in D_i.
- $\mathcal{P}_i$ which is the set of ordered pairs of vertices (a, b) such that: (i) $a, b \in X_i$, and (ii) D_i contains the path $a \rightsquigarrow b$.

- $\mathcal{F}_i$ which is the set of ordered pairs of vertices (a, b) such that: (i) $a, b \in X_i$, and (ii) connecting a to b with a directed path makes a non-admissible edge of D_i to become transitive.
- c_i which is the *cost* of R_i, that is, the number of transitive edges in D_i. Note that $c_i \geq |\mathcal{A}_i|$.
- $\mathcal{S}_i$ which maps each vertex $v \in X_i$ to a Boolean value $\mathcal{S}_i(v)$ that is true if and only if v is a source in D_i. Analogously, $\mathcal{T}_i$ maps each vertex $v \in X_i$ to a Boolean value $\mathcal{T}_i(v)$ that is true if and only if v is a sink in D_i.
- σ_i which is a flag that indicates whether $D_O(G)$ contains a source that belongs to G_i but not to X_i. Analogously, τ_i is a flag that indicates whether $D_O(G)$ contains a sink that belongs to G_i but not to X_i.

Observe that, for a bag X_i, different solutions O and O' of G may be encoded by the same record R_i. In this case, O and O' are *equivalent.* Clearly, this defines an equivalent relation on the set of solutions for G, and each record represents an equivalence class. The goal of the algorithm is to incrementally construct the set of records (i.e., the quotient set) for each bag rather than the whole set of solutions. More formally, for each bag $X_i \in \mathcal{X}$, we associate a set of records $\mathcal{R}_i = \{R_i^1, ..., R_i^h\}$. While this is not essential for establishing fixed-parameter tractability, we further observe that if more records are equal except for their costs, it suffices to keep in $\mathcal{R}_i$ the one whose cost is no larger than any other record. The next lemma easily follows.

▶ **Lemma 1** (⋆). *For a bag X_i, the cardinality of $\mathcal{R}_i$ is $2^{O(\omega^2)}$. Also, each record of $\mathcal{R}_i$ has size $O(\omega^2)$.*

3.2 Description of the algorithm

We are now ready to describe our dynamic-programming algorithm over a nice tree-decomposition $(\mathcal{X}, T)$ of the input graph G. Let X_i be the current bag visited by the algorithm. We compute the records of X_i based on the records computed for its child or children (if any). If the set of records of a bag is empty, the algorithm halts and returns a negative answer. We distinguish four cases based on the type of the bag X_i. Observe that, to index the records within $\mathcal{R}_i$, we added a superscript $q \in [h]$ to each record, and we will do the same for all the record's elements.

X_i is a leaf bag. We have that X_i is the empty set and $\mathcal{R}_i$ consists of only one record, i.e., $\mathcal{R}_i = \{R_i^1 = \langle \emptyset, \emptyset, \emptyset, \emptyset, 0, \emptyset, \emptyset, \text{false}, \text{false} \rangle\}$.

X_i is an introduce bag. Let $X_j = X_i \setminus \{v\}$ be the child of X_i. Initially, $\mathcal{R}_i = \emptyset$. Next, the algorithm exhaustively extends each record $R_j^p \in \mathcal{R}_j$ to a set of records of $\mathcal{R}_i$ as follows. Let $\mathcal{O}_v$ be the set of all the possible orientations of the edges incident to v in $G[X_i]$, and similarly let $\mathcal{A}_v$ be the set of all the possible subsets of the edges incident to v in $G[X_i]$. The algorithm considers all possible pairs (o, t) such that $o \in \mathcal{O}_v$ and $t \in \mathcal{A}_v$. For each pair (o, t), we proceed as follows.

1. Let $q = |\mathcal{R}_i| + 1$, the algorithm computes a candidate orientation $\mathcal{O}_i^q$ of $G[X_i]$ starting from $\mathcal{O}_j^p$ and orienting the edges of v according to o.
2. Similarly, it computes the candidate set of admissible edges $\mathcal{A}_i^q$ starting from $\mathcal{A}_j^p$ and adding to it the edges in t.
3. Next, it sets the candidate cost $c_i^q = c_j^p + |t|$.

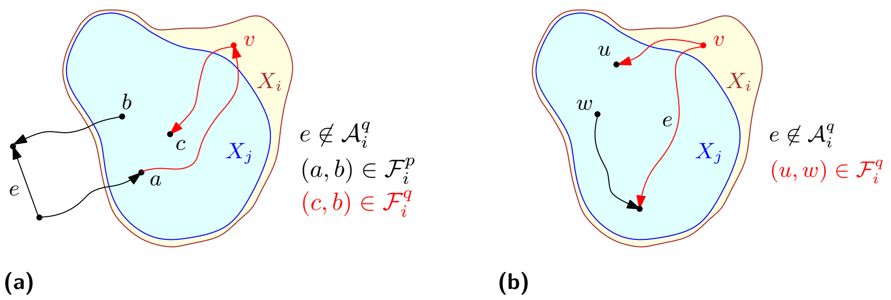

v
X_i
b
c
X_j
a
e
$e \notin \mathcal{A}_i^q$
$(a,b) \in \mathcal{F}_i^p$
$(c,b) \in \mathcal{F}_i^q$
(a)
u
v
X_i
w
e
X_j
$e \notin \mathcal{A}_i^q$
$(u,w) \in \mathcal{F}_i^q$
(b)

Figure 4 Illustration of Step 5c of the algorithm when X_i is an introduce bag.

4. Let the *extension* $\langle \mathcal{O}_i^q, \mathcal{A}_i^q, c_i^q \rangle$ be *valid* if:
   a. $c_i^q \leq k$;
   b. there is no pair $(a,b) \in \mathcal{P}_j^p$ so that $bv, va \in D[X_i^q]$;
   c. there is no pair $(a,b) \in \mathcal{F}_j^p$ so that $av, vb \in D[X_i^q]$.
   Clearly, if an extension is not valid, the corresponding record cannot encode any solution; namely, condition (a) ensures that the candidate cost does not exceed k, condition (b) guarantees the absence of cycles, condition (c) guarantees that no non-admissible edge becomes transitive. Hence, if an extension is not valid, the algorithm discards it and continues with the next pair (o,t).

5. Instead, if the extension is valid, the algorithm computes the record $R_i^q = \langle \mathcal{O}_i^q, \mathcal{A}_i^q, \mathcal{P}_i^q, \mathcal{F}_i^q, c_i^q, \mathcal{S}_i^q, \mathcal{T}_i^q, \sigma_i^q, \tau_i^q \rangle$ of $\mathcal{R}_i$, where $\sigma_i^q = \sigma_j^p$, $\tau_i^q = \tau_j^p$ (recall that $X_j \subset X_i$). To complete the record R_i^q, it remains to compute $\mathcal{S}_i^q$, $\mathcal{T}_i^q$, $\mathcal{P}_i^q$ and $\mathcal{F}_i^q$.
   a. For each vertex $w \in X_j$, we set $\mathcal{S}_i^q(w) =$ true if and only if $\mathcal{S}_j^p(w) =$ true and there is no edge of v oriented from v to w in $D[X_i^q]$ (which would make w not a source anymore). Similarly, for each vertex $w \in X_j$, we set $\mathcal{T}_i^q(w) =$ true if and only if $\mathcal{T}_j^p(w) =$ true and there is no edge of v oriented from w to v in $D[X_i^q]$. Finally, we set $\mathcal{S}_i^q(v) =$ true if and only if v is a source in $D[X_i^q]$ (as by the definition), and we set $\mathcal{T}_i^q(v) =$ true if and only if v is a sink in $D[X_i^q]$.
   b. We initially set $\mathcal{P}_i^q = \emptyset$. We recompute the paths from scratch as follows. We build an auxiliary digraph D^* which we initialize with $D[X_i^q]$. We then add to D^* the information about paths in $\mathcal{P}_j^p$. Namely, for each $(a,b) \in \mathcal{P}_j^p$, we add an edge ab to D^* (if it does not already exists). Once this is done, for each pair $u, w \in X_i \times X_i$ for which there is a path $u \rightsquigarrow w$ in D^*, we add the pair (u,w) to $\mathcal{P}_i^q$.
   c. Consider now $\mathcal{F}_i^q$. We initially set $\mathcal{F}_i^q = \mathcal{F}_j^p$. Observe that the addition of v might have created new pairs of vertices that should belong to $\mathcal{F}_i^q$. Namely, for each pair $(a,b) \in \mathcal{F}_j^p$, we verify what are the vertices c such that $D[X_i^q]$ contains a path $a \rightsquigarrow c$ while $D[X_j^p]$ does not (observe that $a \rightsquigarrow c$ contains v, possibly $c = v$); for each such vertex, we add (c,b) to $\mathcal{F}_i^q$. See Figure 4a for an illustration. Similarly, we verify what are the vertices d such that $D[X_i^q]$ contains a path $d \rightsquigarrow b$ while $D[X_j^p]$ does not (again $d \rightsquigarrow b$ contains v, possibly $d = v$); for each such vertex, we add (a,d) to $\mathcal{F}_i^q$. Finally, we consider all the edges incident to v and that are not in $\mathcal{A}_i^q$. These edges are not admissible and we should further update $\mathcal{F}_i^q$ accordingly. This can be done as follows: we consider each edge incident to v not in $\mathcal{A}_i^q$, for each such an edge e we verify what are the pairs of vertices in X_i (including e's endpoints) such that connecting them with a path makes e transitive, we add such pairs to $\mathcal{F}_i^q$ if not already present. See Figure 4b for an illustration.

X_i is a forget bag. Let $X_j = X_i \cup \{v\}$ be the child of X_i. The algorithm computes $\mathcal{R}_i$ by exhaustively merging records of $\mathcal{R}_j$ as follow.

1. For each $R_j^p \in \mathcal{R}_j$, we remove from $\mathcal{O}_j^p$ and $\mathcal{A}_j^p$ all the edges incident to v and from $\mathcal{P}_j^p$ and $\mathcal{F}_j^p$ all the pairs where one of the vertices is v. Observe that due to this operation, there might now be records that are identical except possibly for their costs. Among them, we only keep one record whose cost is no larger than any other record.
2. Let R_j^p be a record of $\mathcal{R}_j$ that was not discarded by the procedure above. If $\mathcal{S}_j^p(v) \wedge \sigma_j^p$, we discard R_j^p (because the encoded orientation would contain two sources), else we set σ_j^p = true (because v is a source). Similarly, if $\mathcal{T}_j^p(v) \wedge \tau_j^p$, we discard R_j^p, else we set τ_j^p = true. At this point, if the record has not been discarded yet and vertices s and t are prescribed, we can add the following check. If $\mathcal{S}_j^p(v) \wedge \sigma_j^p$, then v is a source, hence if $v \neq s$, we discard the record. Analogously, if $\mathcal{T}_j^p(v) \wedge \tau_j^p$, then v is a sink, hence if $v \neq t$, we discard the record.
3. Finally, we remove from $\mathcal{S}_j^p$ and $\mathcal{T}_j^p$ the values $\mathcal{S}_j^p(v)$ and $\mathcal{T}_j^p(v)$.
4. All the records that have not been discarded and have been updated according to the above procedure are added to $\mathcal{R}_i$.

X_i is a join bag. Let $X_j = X_{j'}$ be the two children of X_i. The algorithm computes $\mathcal{R}_i$ by exhaustively checking if a pair of records, one from X_j and one from $X_{j'}$, can be merged together. For each pair R_j^p and $R_{j'}^{p'}$, we proceed as follows.

1. We initially set $\mathcal{R}_i = \emptyset$. The two records R_j^p and $R_{j'}^{p'}$ are *mergeable* if:
   a. $\mathcal{O}_j^p = \mathcal{O}_{j'}^{p'}$;
   b. $\mathcal{A}_j^p = \mathcal{A}_{j'}^{p'}$;
   c. $c_j^p + c_{j'}^{p'} - |\mathcal{A}_j^p| \leq k$;
   d. there is no pair $(a, b) \in \mathcal{P}_j^p$ such that $(b, a) \in \mathcal{P}_{j'}^{p'}$;
   e. there is no pair $(a, b) \in \mathcal{P}_j^p$ such that $(a, b) \in \mathcal{F}_{j'}^{p'}$;
   f. there is no pair $(a, b) \in \mathcal{P}_{j'}^{p'}$ such that $(a, b) \in \mathcal{F}_j^p$;
   g. $\neg(\sigma_j^p \wedge \sigma_{j'}^{p'})$;
   h. $\neg(\tau_j^p \wedge \tau_{j'}^{p'})$.

   Conditions **a-b** are obviously necessary to merge the records. Condition **c** guarantees that the number of transitive edges (avoiding double counting the admissible edges in X_i) is at most k. Condition **d** guarantees the absence of cycles. Conditions **e-f** guarantee that no non-admissible edge becomes transitive. Conditions **g-h** guarantee that the resulting orientation contains at most one source and one sink. If the two records are not mergeable, we discard the pair and proceed with the next one. Otherwise we create a new record R_i^q, with $q = |\mathcal{R}_i| + 1$, and continue to the next step.
2. Based on the previous discussion, we can now compute R_i^q as follows:
   a. $\mathcal{O}_i^q = \mathcal{O}_j^p$;
   b. $\mathcal{A}_i^q = \mathcal{A}_j^p$;
   c. $c_i^q = c_j^p + c_{j'}^{p'} - |\mathcal{A}_j^p|$;
   d. For each pair (a, b) of vertices of X_i, we add it to $\mathcal{P}_i^q$ if it is contained in $\mathcal{P}_j^p$ or in $\mathcal{P}_{j'}^{p'}$.
   e. For each pair (a, b) of vertices of X_i, we add it to $\mathcal{F}_i^q$ if $\mathcal{F}_j^p(a, b) \vee \mathcal{F}_{j'}^{p'}(a, b)$.
   f. For each vertex v of X_i, we set $\mathcal{S}_i^q(v) = \mathcal{S}_j^p(v) \wedge \mathcal{S}_{j'}^{p'}(v)$;
   g. For each vertex v of X_i, we set $\mathcal{T}_i^q(v) = \mathcal{T}_j^p(v) \wedge \mathcal{T}_{j'}^{p'}(v)$;
   h. $\sigma_i^q = \sigma_j^p \vee \sigma_{j'}^{p'}$;
   i. $\tau_i^q = \tau_j^p \vee \tau_{j'}^{p'}$.

The next lemma establishes the correctness of the algorithm.

▶ **Lemma 2.** *Graph G admits a solution for* ST-ORIENTATION *if and only if the algorithm terminates after visiting the root of T. Also, the algorithm outputs a solution, if any.*

Proof. ($\rightarrow$) Suppose that the algorithm terminates after visiting the root bag X_ρ of T. We reconstruct a solution O of G as follows. We can assume that our algorithm stores additional pointers for each record (a common practice in dynamic programming), such that each record has a single outgoing pointer (and potentially many incoming pointers). Consider a record R_i^q of a bag X_i. If X_i is an introduce bag, there is only one record R_j^p of the child bag X_j from which R_i^q was generated and the pointer links R_i^q and R_j^p. If X_i is forget bag, there might be multiple records that have been merged into R_i^q and in this case the pointer link R_i^q with one of these records with minimum cost. If X_i is a join bag, there are two mergeable records R_j^p and $R_{j'}^{p'}$ that have been merged together, and the pointer links R_i^q to R_j^p and $R_{j'}^{p'}$. With these pointers at hand, we can apply a top-down traversal of T, starting from the single (empty) record of the root bag X_ρ and reconstruct the corresponding orientation O. Namely, when visiting an introduce bag and the corresponding record, we orient the edges of the introduced vertex v according to the orientation O_v defined by the record.

We now claim that $D_O(G)$ is an st-graph with at most k transitive edges. Suppose first, for a contradiction, that $D_O(G)$ contains more than one source. Let s and s' be two sources of $D_O(G)$. Then $\mathcal{S}_i^q(s) =$ false in the bag X_i in which s has been forgotten, and similarly for $\mathcal{S}_i^q(s')$. This is however not possible by construction of $\mathcal{S}_i^q$. Thus, either the record R_i^q has been discarded because $S_j^p(v) \wedge \sigma_j^p$ (see item **2** when X_i is a forget bag) or $\sigma_j^p =$ false. The first case contradicts the fact that R_i^q is a record used to reconstruct O. The second case implies that s' has not been encountered; however, in this latter case the algorithm sets $\sigma_j^p =$ true, hence some descendant record will be discarded as soon as s' is forgotten, again contradicting the fact that we are considering records with pointers up to the root bag. A symmetric argument shows that $D_O(G)$ contains a single sink. We next argue that $D_O(G)$ is acyclic. Suppose, again for a contradiction, that $D_O(G)$ contains a cycle. In particular, the cycle was created either in an introduce bag or in a join bag. In the former case, let v be the last vertex of this cycle that has been introduced in a bag X_i. Let a, b be the neighbors of v that are part of the cycle, and w.l.o.g. assume that the edges are va and bv. It must be $\mathcal{P}_i^q$ does not contain the pair (a, b), otherwise we would have discarded this particular orientation for the edges incident to v (see item **4.b** when X_i is an introduce bag). On the other hand, one easily verifies that when introducing a vertex v, all the new paths involving v are computed from scratch (see item **5.b** when X_i is an introduce bag), and, similarly, when joining two bags, the existence of a path in one of the two bags is correctly reported in the new record (see item **2.d** when X_i is a join bag). If the cycle was created in a join bag the argument is analogous, in particular, observe that we verify that there is no path contained in the record of one of the child bags such that the same path with reversed direction exists in the record of the other child bag (see item **1.d** when X_i is a join bag). We conclude this direction of the proof by showing that $D_O(G)$ contains at most k transitive edges. Observe first that the cost of the record ensures that at most k edges of G are part of some set of admissible edges. Suppose, for a contradiction, that $D_O(G)$ contains more than k transitive edges. Then there is a bag X_i and a record R_i^q in which a non-admissible edge became transitive. Also, X_i is either an introduce or a join bag. If X_i introduced a vertex v, observe that all the newly introduced edges are incident to v. On the other hand, the algorithm discarded the orientations of the edges of v for which there is a pair $(a, b) \in \mathcal{F}_j^p$ (with X_j being the child of X_i) so that $av, vb \in D[X_i^q]$ (see item **4.c** when

X_i is an introduce bag). Then either the orientation was discarded, which contradicts the fact that we are considering a record used to build the solution, or $\mathcal{F}_j^p$ missed the pair (a, b). Again one verifies this second case is not possible, because the new pairs that are formed in an introduce bag are correctly identified (see item **5.c** when X_i is an introduce bag) by the algorithm and similarly for join bags (see item **2.e** when X_i is a join bag). If X_i is a join bag, the argument is analogous, in particular, we verified that there is no path in one of the two child records that makes transitive a non-admissible edge in the other child record (see items **1.e** and **1.f** when X_i is a join bag). This concludes the first part of the proof.

($\leftarrow$) It remains to prove that, if G admits a solution O, then the algorithm terminates after visiting the root X_ρ of T. If this were not the case, there would be a bag X_i of T and a candidate record that encodes O, such that the record has been incorrectly discarded by the algorithm; we show that this is not possible. Suppose first that X_i is an introduce bag. Then a candidate record is discarded if the cost exceeds k, or if a cycle is created, or if a non-admissible edge becomes transitive (see the conditions of item **4** when X_i is an introduce bag). In all cases the candidate record does not encode a solution. If X_i is a forget bag, we may discard a candidate record if it is identical to another but has a non-smaller cost (see item **1** when X_i is a forget bag). Hence we always keep a record that either encodes the solution at hand or a solution with fewer transitive edges but with exactly the same interface at X_i. Also, we may discard a record if the forgotten vertex v is a source and G_i already contains a source (see item **2** when X_i is a forget bag). This is correct, because no further edge can be added to v after it is forgotten. A symmetric argument holds for the case in which a record is discarded due to v being a sink. Finally, if X_i is a join bag, pairs of records of its children bags are discarded if not mergeable (see the conditions of item **1** when X_i is a join bag). One easily verifies that failing one of the conditions for mergeability implies that the record does not encode a solution (see also the discussion after item **1**). ◀

The next theorem summarizes our contribution.

▶ **Theorem 3** ($\star$). *Given an input graph $G = (V, E)$ of treewidth ω and an integer $k \geq 0$, there is an algorithm that either finds a solution of* ST-ORIENTATION *or reject the input in time $2^{O(\omega^2)} \cdot n$.*

4 The Complexity of the Non-Transitive st-OrientationProblem for Graphs of Bounded Diameter and Bounded Degree

We begin by recalling the special case of ST-ORIENTATION considered in [1]. An *st*-orientation O of a graph G is *non-transitive* if $D_O(G)$ does not contain transitive edges.

NON-TRANSITIVE ST-ORIENTATION (NT-ST-ORIENTATION)
Input: An undirected graph $G = (V, E)$, and two vertices $s, t \in V$.
Output: An non-transitive *st*-orientation O of G such that vertices s and t are the source and sink of $D_O(G)$, respectively.

The hardness proof of NT-ST-ORIENTATION in [1] exploits a reduction from NOT-ALL-EQUAL 3-SAT (NAE-3-SAT) [13]. Recall that the input of NAE-3-SAT is a pair $\langle X, \varphi\rangle$ where X is a set of boolean variables and φ is a set of clauses, each composed of three literals out of X, and the problem asks for an assignment of the variables in X so that each clause in φ is composed of at least one true variable and one false variable.

In this section, we show that NT-ST-ORIENTATION is NP-hard even for graphs of bounded diameter and for graphs of bounded vertex degree that are subdivisions of triconnected graphs. To prove our results, we first summarize the construction used in [1].

4.1 A Glimpse into the Hardness Proof of NT-st-Orientation

The construction in [1] adopts three types of gadgets, which we recall below. Given an edge e of a digraph D such that e has an end-vertex v of degree 1, we say that e *enters* D if it is outgoing with respect to v, and we say that e *exits* D otherwise. Similarly, given a directed edge $e = uv$, we say that e *exits* u and that e *enters* v.

- The *fork gadget* F is depicted in Figure 5a. See Lemma 1 of [1]. Namely, if F does not contain s or t (the source and sink prescribed in the input), then in any non-transitive orientation O of a graph G containing F, either e_1 enters F and e_9, e_{10} exit F, or vice versa. Figure 5a depicts F, $D_{O_1}(F)$ and $D_{O_2}(F)$, where O_1 and O_2 are the two st-orientations admitted by F.
- The *variable gadget* G_x associated to a variable $x \in X$ is shown in Figure 5b; observe it contains the designated vertices s and t. Its crucial property is stated in Lemma 2 of [1]. Namely, in any non-transitive st-orientation O of a graph G containing G_x, either x exists G_x and $\overline{x}$ enters G_x, or vice-versa.
- The *split gadget* S_k is shown in Figure 5c; it consists of $k-1$ fork gadgets chained together, for some fixed $k > 0$. The crucial property of this gadget is described in Lemma 3 of [1]. Namely, in any non-transitive st-orientation O of a graph G containing S_k, either x (the *input edge* of S_k) enters S_k and the edges e_9 and e_{10} of the fork gadgets $F_1, \dots, F_{k-1}$ incident to one degree-1 vertex (the *outgoing edges* of S_k) exit S_k, or vice-versa.

Given an instance $\langle X, \varphi \rangle$ of NAE-3-SAT, the instance $\langle G_\varphi, s, t \rangle$ of NT-ST-ORIENTATION is constructed as follow. For each $x \in X$ we add G_x and two split gadgets S_k and $S_{\overline{k}}$, where k (resp. $\overline{k}$) is the number of clauses where x appears in its non-negated (resp. negated) form (edges x and $\overline{x}$ are the input edges of S_k and $S_{\overline{k}}$, respectively). Finally, for each clause $c = (x_1, x_2, x_3) \in \varphi$, we add a vertex c that is incident to an output edge of the split gadget of each of its variables. See Figure 6b, where the non-dashed edges and all the vertices with the exception of g define G_φ. It can be shown that $\langle X, \varphi \rangle$ is a yes-instance of NAE-3-SAT if and only if $\langle G_\varphi, s, t \rangle$ is a yes-instance of NT-ST-ORIENTATION [1].

4.2 Hardness for Graphs of Bounded Diameter

Given an undirected graph G, the *distance* between two vertices of G is the length of any shortest path connecting them. The *diameter* of G is the maximum distance over all pairs of vertices of the graph. We now adapt the construction in Section 4.1 to show that NT-ST-ORIENTATION remains NP-hard also for graphs of bounded diameter. We define the *extended fork gadget* by adding an edge e_{11} to the fork gadget (see Figure 6a).

Construction of H_φ. Given an instance $\langle X, \varphi \rangle$ of NAE-3-SAT and the instance $\langle G_\varphi, s, t \rangle$ of NT-ST-ORIENTATION computed as described in Section 4.1, we define $\langle H_\varphi, s, t \rangle$ as follows. We first set $H_\varphi = G_\varphi$. Then, we add a vertex g to H_φ and an edge (g, f) for each vertex f belonging to a fork F of H_φ and incident to the corresponding edges e_3, e_6, and e_7. Also, we add edges (g, t) and (s, g), and we subdivide each of them once. See Figure 6b (the non-dashed edges and all the vertices with the exception of g define G_φ).

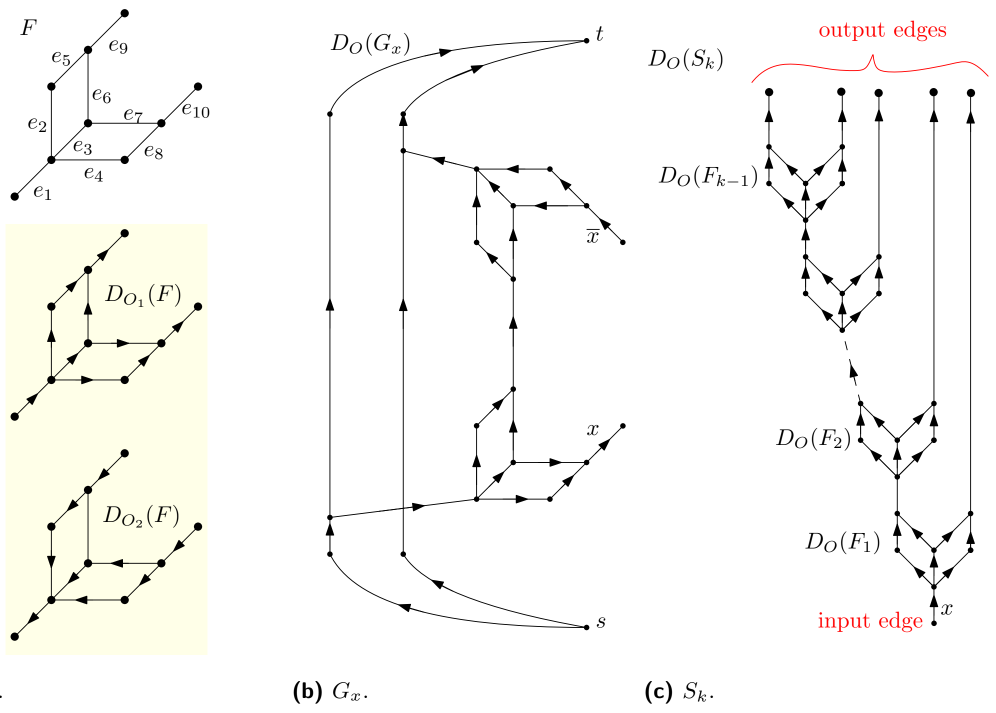


Figure 5 (a) The fork gadget F and its two possible non-transitive *st*-orientations. (b) The variable gadget $D_O(G_x)$ associated to $x \in X$, where O is one of its two possible orientations. (c) The split gadget $D_O(S_k)$ associated to x, where O is one of its two possible orientations.

▶ **Theorem 4** ($\star$). *NT-ST-ORIENTATION is* NP*-hard for graphs of diameter at most* 6.

Proof sketch. We construct H_φ as described above. Observe that any vertex of G is at distance at most 3 to g, hence H_φ has diameter at most 6. We show that a non-transitive *st*-orientation of G_φ corresponds to a non-transitive *st*-orientation of H_φ ($\rightarrow$) and vice versa ($\leftarrow$).

($\rightarrow$) Given a non-transitive *st*-orientation O' of G_φ, we construct an *st*-orientation O of H_φ by extending O' as follow. We orient the four edges of $H_\varphi \setminus G_\varphi$ connecting s to t such that the path is directed from s to g. For each other edge e, which is incident to g, we orient it so that e enters g if and only if e is the edge incident to an extended fork gadget whose corresponding edge e_1 is an entering edge. See Figure 6b. For each two vertices $a, b \in D_O(H_\varphi)$, there is no path $a \rightsquigarrow b$ so that $ag, gb \in D_O(H_\varphi)$. Hence, since $D_{O'}(G_\varphi)$ has no cycle, also $D_O(H_\varphi)$ has no cycle. Consequently, O is an acyclic orientation with s and t being its single source and sink, respectively. We now show that it does not contain transitive edges. Let $e = ab$ be any edge of $D_O(H_\varphi)$. We have that any path from $s \rightsquigarrow t$ containing g either contains edges incident to degree-2 vertices or edges e_1, e_3, and e_{11} of an extended fork gadget. All these edges have endpoints which are not adjacent by construction. Hence, there is no path $a \rightsquigarrow b$ containing g and, since O' is non-transitive, e is not transitive in $D_O(H_\varphi)$.

($\leftarrow$) This direction is based on the observation that, given an extended fork gadget F, in any non-transitive *st*-orientation O of G, either e_3 enters f and e_6, e_7 exit f or vice versa. ◀

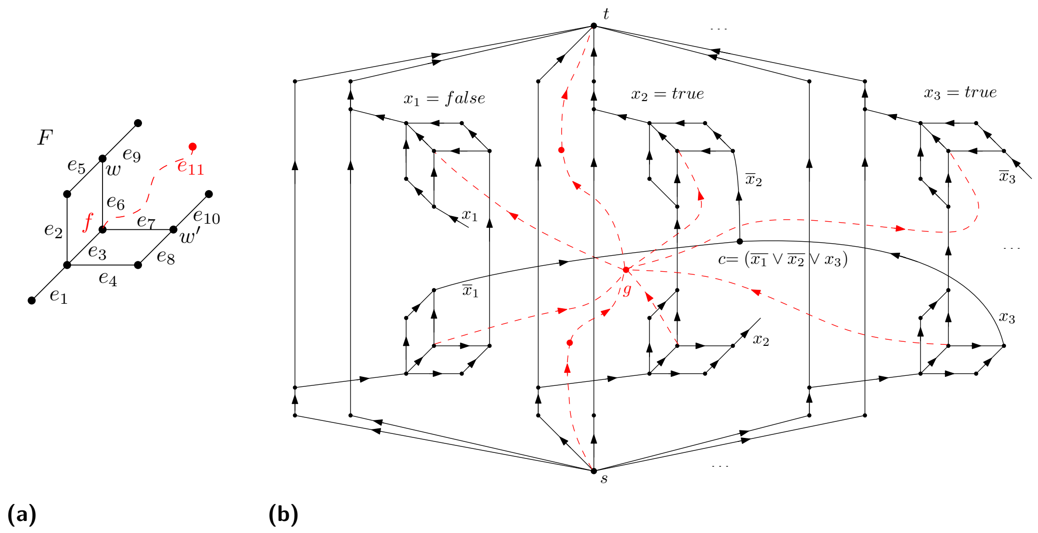


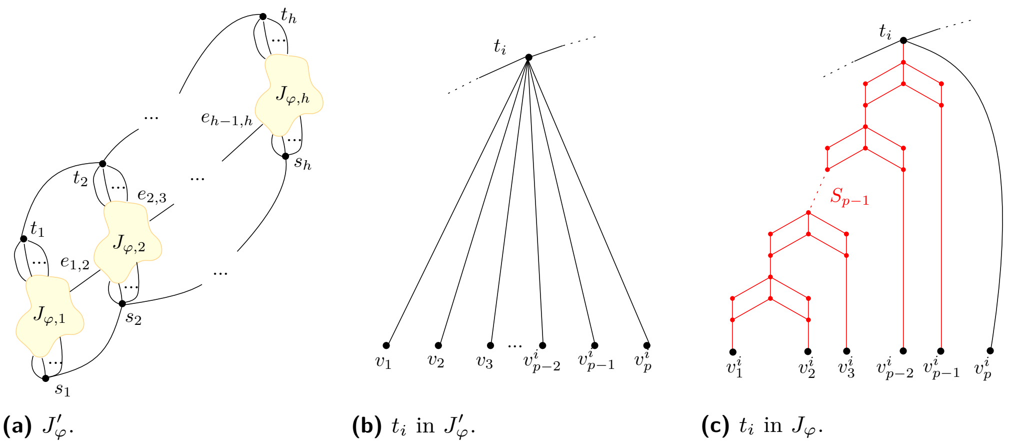
Figure 6 (a) A fork gadget F extended with edge e_{11}. (b) Graphs $D_{O'}(G_\varphi)$, defined by the non-dashed edges, and graph $D_O(H_\varphi)$, obtained from G by adding g and the dashed edges. O' and O are non-transitive st-orientations of G_φ and H_φ, respectively. O is obtained by extending O'.

4.3 Hardness Subdivisions of Triconnected Graphs with Bounded Degree

We prove now that NT-ST-ORIENTATION is NP-hard even if G is a *4-graph*, i.e., the degree of each vertex is at most 4, and, in addition, it is a subdivision of a triconnected graph.

Construction of J_φ. Given an instance $\langle X, \varphi\rangle$ of NAE-3-SAT and the instance $\langle G_\varphi, s, t\rangle$ of NT-ST-ORIENTATION computed as described in Section 4.1, we compute $\langle J_\varphi, s, t\rangle$ as follows. We remove s and t from $J''_\varphi = G_\varphi$. We obtain a disconnected graph whose connected components are $J_{\varphi,1}, ..., J_{\varphi,h}$. We add a vertex s_i and a vertex t_i to each $J_{\varphi,i}$ (which will play the role of local sources and sinks for each component). Next, for each $i \in [1, h-1]$: (i) We add the edge (s_i, s_{i+1}) and (t_{i+1}, t_i); (ii) We add an edge $e_{i+1,i}$ incident to a vertex identified as the f-vertex of a fork gadget of $J_{\varphi,i+1}$ and to a vertex identified as the f-vertex of a fork gadget of $J_{\varphi,j}$. We denote by J'_φ the obtained graph; see Figure 7a for a schematic illustration. For each $J_{\varphi,i}$ ($i \in [1, h]$) of J'_φ, the only vertices having degree higher than 4 are s_i and t_i. For each $i \in [1, h]$, we proceed as follow. We first consider t_i. If t_i has degree $p \le 4$, we do nothing. Otherwise, if $p \ge 5$, we proceed as follows: (i) We consider $p-1$ edges incident to t_i and to vertices of $J_{\varphi,i}$ and we remove them; (ii) We connect the endpoints $v^i_1, ..., v^i_p$ of the removed edges that are not t_i to a split gadget S_{p-1} and t_i to the input edge of S_{p-1}. Figure 7b depicts t_i ($i \in [1, h]$) and its neighborhood $\{v^i_1, ..., v^i_p\}$ in J'_φ and Figure 7c depicts how t_i is connected to the vertices $v^i_1, ..., v^i_p$ after the above operation. We perform a symmetric operation on s_i. The resulting graph is denoted by J_φ, and it has vertex degree at most four by construction.

▶ **Theorem 5** ($\star$). *NT-ST-ORIENTATION is* NP*-hard for 4-graphs that are subdivisions of triconnected graphs.*



t_h
$J_{\varphi,h}$
$e_{h-1,h}$
s_h
t_2
$e_{2,3}$
t_1
$J_{\varphi,2}$
$e_{1,2}$
$J_{\varphi,1}$
s_2
s_1
(a) J'_φ.
t_i
v_1 v_2 v_3 v^i_{p-2} v^i_{p-1} v^i_p
(b) t_i in J'_φ.
t_i
S_{p-1}
v^i_1 v^i_2 v^i_3 v^i_{p-2} v^i_{p-1} v^i_p
(c) t_i in J_φ.

Figure 7 (a) Schematic representation of graph J'_φ. (b-c) How vertex t_i is connected to its neighbourhood in (b) J'_φ and (c) J_φ.

5 Open Problems

Several interesting open problems stem from our research. Among them:

- Is there an FPT-algorithm for the ST-ORIENTATION problem parameterized by treewidth running in $2^{o(\omega^2)} \cdot \text{poly}(n)$ time?
- Does ST-ORIENTATION parameterized by treedepth admit a polynomial kernel?
- We have shown that finding non-transitive *st*-orientations is NP-hard for graphs of vertex degree at most four. On the other hand, the problem is trivial for graphs of vertex degree at most two. What is the complexity of the problem for vertex degree at most three? Similarly, one can observe that the problem is easy for graphs of diameter at most two, while it remains open the complexity for diameter in the range $[3, 5]$.

References

1 Carla Binucci, Walter Didimo, and Maurizio Patrignani. *st*-orientations with few transitive edges. In Patrizio Angelini and Reinhard von Hanxleden, editors, *GD 2022*, volume 13764 of *LNCS*, pages 201–216. Springer, 2022. doi:10.1007/978-3-031-22203-0_15.

2 Carla Binucci, Giuseppe Liotta, Fabrizio Montecchiani, Giacomo Ortali, and Tommaso Piselli. On the parameterized complexity of computing *st*-orientations with few transitive edges. *CoRR*, abs/2306.03196, 2023. arXiv:2306.03196.

3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

4 Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996. doi:10.1006/jagm.1996.0049.

5 Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

6 Giuseppe Di Battista and Roberto Tamassia. Algorithms for plane representations of acyclic digraphs. *Theor. Comput. Sci.*, 61:175–198, 1988. doi:10.1016/0304-3975(88)90123-5.

7 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.

8 Shimon Even and Robert Endre Tarjan. Computing an st-numbering. *Theoretical Computer Science*, 2(3):339–344, 1976. doi:10.1016/0304-3975(76)90086-4.

9 Michael Kaufmann and Dorothea Wagner, editors. *Drawing Graphs, Methods and Models*, volume 2025 of *LNCS*. Springer, 2001. doi:10.1007/3-540-44969-8.

10 Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *LNCS*. Springer, 1994.

11 A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In *Theory of Graphs: International Symposium.*, pages 215–232. Gorden and Breach, 1967.

12 Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

13 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing (STOC 1978)*, pages 216–226. Association for Computing Machinery, 1978. doi:10.1145/800133.804350.