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

Carla Binucci, Giuseppe Liotta, Fabrizio Montecchiani,
Giacomo Ortali, and Tommaso Piselli

Dipartimento di Ingegneria, Università degli Studi di Perugia, Italy
name.surname@unipg.it

Abstract. We study the problem of computing an *st*-orientation of a graph with at most k transitive edges, which 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.

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). In particular, we consider constrained acyclic orientations, which find applications in several domains, including graph planarity and graph drawing [1,7,9]. More specifically, given a graph $G = (V, E)$ and two vertices $s, t \in V$, an *st-orientation* of G, also known as *bipolar 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 at the core of planarity testing algorithms [10,13]. In the field of graph drawing, bipolar orientations are a central algorithmic tool to compute different types of layouts (see [5,11] for references). On a similar note, a prominent 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 [2] 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 Fig. 1c are transitive. Besides being of theoretical interest, such orientations, when they exist, can be used to compute readable and compact drawings of graphs [2]; see Fig. 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. [2] prove that the

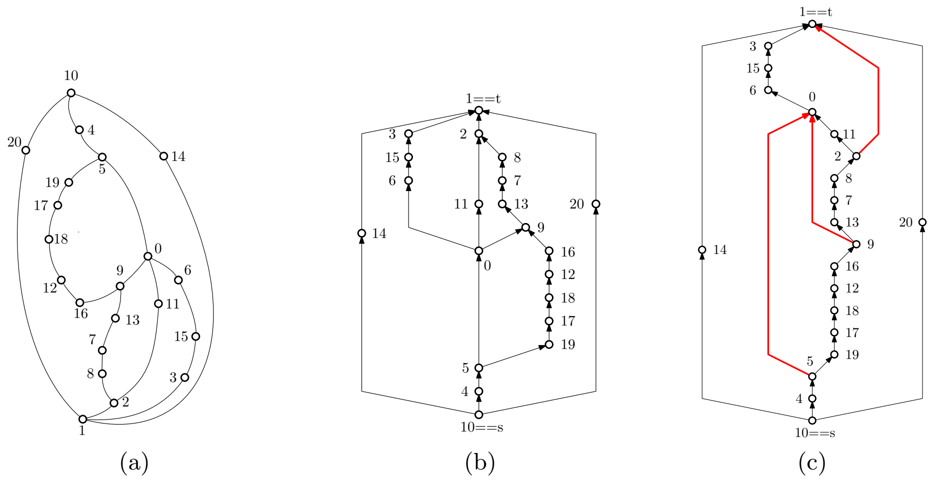

Fig. 1: (a): An undirected graph G. (b)-(c): Two polyline drawings of G computed by using different st-orientations. The drawing in (b) uses an st-orientation without transitive edges and it has smaller area and number of bends than the drawing in (c).

problem of deciding whether an st-orientation with no transitive edges exists is NP-complete.

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. This problem is para-NP-hard by the natural parameter k [2]. 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 [8,14]). 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, our approach avoids storing information about all edges that may potentially become transitive; instead, it 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.
For space reasons, many proofs have been omitted, see [3] for the full version.

Notation. 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 $k \geq 1$.
Output: An *st*-orientation O of G such that the resulting digraph $D_O(G)$ contains at most k transitive edges.

In what follows, for a graph $G = (V, E)$, the pair $(\mathcal{X}, T)$ denotes a *nice tree-decomposition* of G, such that $\mathcal{X} = \{X_i\}_{i \in [\ell]}$ is a collection of subsets of vertices of G, called *bags*, and T is a tree whose nodes are in one-to-one correspondence with the elements of $\mathcal{X}$. 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 (not necessarily nice) tree-decompositions of G. Refer to [4,12] for the required background.

2 st-Orientation Parameterized by Treewidth

We provide 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 nice tree-decomposition of G of width ω. For a bag $X_i \in \mathcal{X}$, we denote by $G[X_i]$ the 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.

Encoding solutions. Let O be a solution and consider a bag X_i. The *record* R_i of X_i that *encodes* O is 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.

- $\mathcal{O}_i$ is the orientation of $D[X_i]$.
- $\mathcal{A}_i$ 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.*
- $\mathcal{P}_i$ 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 $a \rightsquigarrow b$.

– $\mathcal{F}_i$ 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.
– $c_i \geq |\mathcal{A}_i|$ is the *cost* of R_i, i.e., the number of transitive edges in D_i.
– $\mathcal{S}_i$ 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 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 different solutions O and O' of G may be encoded by the same record R_i, we call O and O' *equivalent*. This defines an equivalent relation on the set of solutions for G, and each record represents an equivalence class. The algorithm incrementally constructs 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 $\mathcal{R}_i = \{R_i^1, ..., R_i^h\}$. The next lemma follows.

Lemma 1. *The cardinality of $\mathcal{R}_i$ is $2^{O(\omega^2)}$, and each record has size $O(\omega^2)$.*

Informal description of the algorithm. 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. To begin with, if 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 follow. 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 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 $c_i^q = c_j^p + |t|$.
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 $(a, b) \in F_j^p$ so that $av, vb \in D[X_i^q]$. Clearly, if an extension is not valid, the corresponding record cannot encode any solution, hence the algorithm discards the extension and continues with the next pair (o, t). Otherwise, we compute 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 $\sigma_i^q = \sigma_j^p$, $\tau_i^q = \tau_j^p$.

5. To complete the record R_i^q, it remains to compute $\mathcal{S}_i^q$, $\mathcal{T}_i^q$, $\mathcal{P}_i^q$ and $\mathcal{F}_i^q$. We omit a formal description. At high level, $\mathcal{S}_i^q$ and $\mathcal{T}_i^q$ can be easily computed by inspecting the vertices in X_j and v. For $\mathcal{P}_i^q$, we simply recompute the paths from scratch. For $\mathcal{F}_i^q$, we observe that the addition of v might have created new pairs of vertices that should belong to $\mathcal{F}_i^q$, hence we identify such pairs and compute $\mathcal{F}_i^q$ accordingly.

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. Due to this operation, there may be records that are identical except possibly for their costs, we only keep the one 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 orientation would contain two sources), else we set $\sigma_j^p = \text{true}$ (because v is a source). Similarly, if $\mathcal{T}_j^p(v) \wedge \tau_j^p$, we discard R_j^p, else we set $\tau_j^p = \text{true}$. 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. Next, we remove from $\mathcal{S}_j^p$ and $\mathcal{T}_j^p$ the values $\mathcal{S}_j^p(v)$ and $\mathcal{T}_j^p(v)$.
3. 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.** This case is omitted for space reasons.

It is possible to prove that graph G admits a solution for ST-ORIENTATION if and only if the algorithm terminates after visiting the root of T.

Theorem 1. *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 $2^{O(\omega^2)} \cdot n$.*

3 Hardness of Non-Transitive st-Orientation

We first recall the special case of ST-ORIENTATION studied in [2]. 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.

In the hardness proof of NT-ST-ORIENTATION in [2], the diameter is unbounded. The authors exploit a reduction from NOT-ALL-EQUAL 3-SAT (NAE-3-SAT) [15]. The construction in [2] adopts three types of gadgets, which we carefully modify to prove the following theorem.

Theorem 2. NT-st-Orientation *is* NP*-hard for graphs of diameter at most* 6 *and for graphs of vertex degree at most* 4*.*

References

1. Angelini, P., Cittadini, L., Di Battista, G., Didimo, W., Frati, F., Kaufmann, M., Antonios, S.: On the perspectives opened by right angle crossing drawings. Journal of Graph Algorithms and Applications **15**(1), 53–78 (2011)
2. Binucci, C., Didimo, W., Patrignani, M.: *st*-orientations with few transitive edges. In: Angelini, P., von Hanxleden, R. (eds.) GD 2022. LNCS, vol. 13764, pp. 201–216. Springer (2022). `https://doi.org/10.1007/978-3-031-22203-0\_15`
3. Binucci, C., Liotta, G., Montecchiani, F., Ortali, G., Piselli, T.: On the parameterized complexity of computing *st*-orientations with few transitive edges. CoRR **abs/2306.03196** (2023)
4. Bodlaender, H.L., Kloks, T.: Efficient and constructive algorithms for the pathwidth and treewidth of graphs. J. Algorithms **21**(2), 358–402 (1996). `https://doi.org/10.1006/jagm.1996.0049`
5. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice-Hall (1999)
6. Di Battista, G., Tamassia, R.: Algorithms for plane representations of acyclic digraphs. Theor. Comput. Sci. **61**, 175–198 (1988). `https://doi.org/10.1016/0304-3975(88)90123-5`
7. Didimo, W., Eades, P., Liotta, G.: Drawing graphs with right angle crossings. In: Dehne, F.K.H.A., Gavrilova, M.L., Sack, J., Tóth, C.D. (eds.) Algorithms and Data Structures, 11th International Symposium, WADS 2009, Banff, Canada, August 21-23, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5664, pp. 206–217. Springer (2009). `https://doi.org/10.1007/978-3-642-03367-4\_19`
8. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Monographs in Computer Science, Springer (1999)
9. Eades, P., Symvonis, A., Whitesides, S.: Three-dimensional orthogonal graph drawing algorithms. Discret. Appl. Math. **103**(1-3), 55–87 (2000). `https://doi.org/10.1016/S0166-218X(00)00172-4`
10. Even, S., Tarjan, R.E.: Computing an st-numbering. Theoretical Computer Science **2**(3), 339–344 (1976). `https://doi.org/https://doi.org/10.1016/0304-3975(76)90086-4`
11. Kaufmann, M., Wagner, D. (eds.): Drawing Graphs, Methods and Models, LNCS, vol. 2025. Springer (2001). `https://doi.org/10.1007/3-540-44969-8`
12. Kloks, T.: Treewidth, Computations and Approximations, LNCS, vol. 842. Springer (1994)
13. Lempel, A., Even, S., Cederbaum, I.: An algorithm for planarity testing of graphs. In: Theory of Graphs: International Symposium. pp. 215–232. Gorden and Breach (1967)
14. Robertson, N., Seymour, P.D.: Graph minors. II. Algorithmic aspects of tree-width. J. Algorithms **7**(3), 309–322 (1986)
15. Schaefer, T.J.: The complexity of satisfiability problems. In: Proceedings of the Tenth Annual ACM Symposium on Theory of Computing (STOC 1978). p. 216–226. Association for Computing Machinery (1978). `https://doi.org/10.1145/800133.804350`