

\textbf{st-Orientations with Few Transitive Edges}\textsuperscript{*}

Carla Binucci\textsuperscript{1}, Walter Didimo\textsuperscript{1}, and Maurizio Patrignani\textsuperscript{2}

\textsuperscript{1} Università degli Studi di Perugia, Italy
\texttt{\{carla.binucci,walter.didimo\}@unipg.it}
\textsuperscript{2} Roma Tre University, Rome, Italy
\texttt{maurizio.patrignani@uniroma3.it}

\textbf{Abstract.} The problem of orienting the edges of an undirected graph such that the resulting digraph is acyclic and has a single source \(s\) and a single sink \(t\) has a long tradition in graph theory and is central to many graph drawing algorithms. Such an orientation is called an \(st\)-orientation. We address the problem of computing \(st\)-orientations of undirected graphs with the minimum number of transitive edges. We prove that the problem is NP-hard in the general case. For planar graphs we describe an ILP model that is fast in practice. We experimentally show that optimum solutions dramatically reduce the number of transitive edges with respect to unconstrained \(st\)-orientations computed via classical \(st\)-numbering algorithms. Moreover, focusing on popular graph drawing algorithms that apply an \(st\)-orientation as a preliminary step, we show that reducing the number of transitive edges leads to drawings that are much more compact.

\section{Introduction}

The problem of orienting the edges of an undirected graph in such a way that the resulting digraph satisfies specific properties has a long tradition in graph theory and represents a preliminary step of several graph drawing algorithms. For example, Eulerian orientations require that each vertex gets equal in-degree and out-degree; they are used to compute 3D orthogonal graph drawings \cite{16} and right-angle-crossing drawings \cite{2}. Acyclic orientations require that the resulting digraph does not contain directed cycles (i.e., it is a DAG); they can be used as a preliminary step to compute hierarchical and upward drawings that nicely represent an undirected graph, or a partially directed graph, so that all its edges monotonically flow in the same direction \cite{4,5,14,17,21,23}.

Specific types of acyclic orientations that are central to many graph algorithms and applications are the so called \textit{st-orientations}, also known as \textit{bipolar orientations} \cite{32}, whose resulting digraphs have a single source \(s\) and a single sink \(t\). It is well known that an undirected graph \(G\) with prescribed vertices \(s\) and \(t\) can be oriented in such a way that the resulting digraph satisfies certain properties.

\textsuperscript{*} Work partially supported by: (i) MIUR, grant 20174LF3T8 AHeAD: efficient Algorithms for HArnessing networked Data”, (ii) Dipartimento di Ingegneria, Università degli Studi di Perugia, grant RICBA21LG: Algoritmi, modelli e sistemi per la rappresentazione visuale di reti.

and \( t \) admits an \( st \)-orientation if and only if \( G \) with the addition of the edge \((s, t)\) (if not already present) is biconnected. The digraph resulting from an \( st \)-orientation is also called an \emph{\( st \)-graph}. An \( st \)-orientation can be computed in linear time via an \( st \)-numbering (or \( st \)-ordering) of the vertices of \( G \) \cite{6,19}, by orienting each edge from the end-vertex with smaller number to the end-vertex with larger number \cite{6}. In particular, if \( G \) is planar, a \emph{planar \( st \)-orientation} of \( G \) additionally requires that \( s \) and \( t \) belong to the external face in some planar embedding of the graph. Planar \( st \)-orientations were originally introduced in the context of an early planarity testing algorithm \cite{26}, and are largely used in graph drawing to compute different types of layouts, including visibility representations, polyline drawings, dominance drawings, and orthogonal drawings (refer to \cite{9,25}). Planar \( st \)-orientations and related graph layout algorithms are at the heart of several graph drawing libraries and software (see, e.g., \cite{7,8,24,34}). Algorithms that compute \( st \)-orientations with specific characteristics (such as bounds on the length of the longest path) are also proposed and experimented in the context of visibility and orthogonal drawings \cite{29,30}.

Our paper focuses on the computation of \( st \)-orientations with a specific property, namely we address the following problem: “Given an undirected graph \( G \)
and two prescribed vertices $s$ and $t$ for which $G \cup (s, t)$ is biconnected, compute an $st$-orientation of $G$ such that the resulting $st$-graph $G'$ has the minimum number of transitive edges (possibly none). We recall that an edge $(u, v)$ of a digraph $G'$ is transitive if there exists a directed path from $u$ to $v$ in $G' \setminus (u, v)$. An $st$-orientation is non-transitive if the resulting digraph has no transitive edges; $st$-graphs with no transitive edges are also known as transitively reduced $st$-graphs [9, 18], bipolar posets [22], or Hasse diagrams of lattices [10, 31]. The problem we study, besides being of theoretical interest, has several practical motivations in graph drawing. We mention some of them:

- Planar $st$-oriented graphs without transitive edges admit compact dominance drawings with straight-line edges, a type of upward drawings that can be computed in linear time with very simple algorithms [11]; when a transitive edge is present, one can temporarily subdivide it with a dummy vertex, which will correspond to an edge bend in the final layout. Hence, having few transitive edges helps to reduce bends in a dominance drawing.

- As previously mentioned, many layout algorithms for undirected planar graphs rely on a preliminary computation of an $st$-orientation of the input graph. We preliminary observed that reducing the number of transitive edges in such an orientation has typically a positive impact on the readability of the layout. Indeed, transitive edges often result in long curves; avoiding them produces faces where the lengths of the left and right paths are more balanced and leads to more compact drawings (see Fig. 1).

- Algorithms for computing upward confluent drawings of transitively reduced DAGs are studied in [18]. Confluent drawings exploit edge bundling to create “planar” layouts of non-planar graphs, without introducing ambiguity [13]. These algorithms can be applied to draw undirected graphs that have been previously $st$-oriented without transitive edges when possible.

We also mention algorithms that compute two-page book embeddings of two-terminal series-parallel digraphs, which either assume the absence of transitive edges [1] or which are easier to implement if transitive edges are not present [12].

**Contribution.** In this paper we first prove that deciding whether a graph admits an $st$-orientation without transitive edges is NP-complete. This is in contrast with the tractability of a problem that is at the opposite of ours, namely, deciding whether an undirected graph has an orientation such that the resulting digraph is its own transitive closure; this problem can be solved in linear time [27].

From a practical point of view, we provide an Integer Linear Programming (ILP) model for planar graphs, whose solution is an $st$-orientation with the minimum number of transitive edges. In our setting, $s$ and $t$ are two prescribed vertices that belong to the same face of the input graph in at least one of its planar embeddings. We prove that the ILP model works very fast in practice. Popular solvers such as CPLEX can find a solution in few seconds for graphs up to 1000 vertices and the resulting $st$-orientations save on average 35% of transitive edges (with improvements larger than 80% on some instances) with respect to applying classical unconstrained $st$-orientation algorithms. Moreover,
focusing on popular graph drawing algorithms that apply an \(st\)-orientation as a preliminary step, we show that reducing the number of transitive edges leads to drawings that are much more compact.

For space restrictions, some details are omitted. Full proofs and additional material can be found in Appendix A.

2 NP-Completeness of the General Problem

We prove that given an undirected graph \(G = (V, E)\) and two vertices \(s, t \in V\), it is NP-complete to decide whether there exists a non-transitive \(st\)-orientation of \(G\). We call this problem Non-Transitive \(st\)-Orientation (NTO). To prove the hardness of NTO we describe a reduction from the NP-complete problem Not-All-Equal 3SAT (NAE3SAT) \cite{33}, where one has a collection of clauses, each composed of three literals out of a set \(X\) of Boolean variables, and is asked to determine whether there exists a truth assignment to the variables in \(X\) so that each clause has at least one true and one false literal.

Starting from a NAE3SAT instance \(\varphi\), we construct an instance \(I_\varphi = \langle G, s, t \rangle\) of NTO such that \(I_\varphi\) is a yes instance of NAE3SAT if and only if \(\varphi\) is a yes instance of NTO. Instance \(I_\varphi\) has one variable gadget \(V_x\) for each Boolean variable \(x\) and one clause gadget \(C_c\) for each clause \(c\) of \(\varphi\). By means of a split gadget, the truth value encoded by each variable gadget \(V_x\) is transferred to all the clause gadgets containing either the direct literal \(x\) or its negation \(\overline{x}\).

Observe that the NAE3SAT instance is in general not “planar”, in the sense that if you construct a graph where each variable \(x\) and each clause \(c\) is a vertex and there is an edge between \(x\) and \(c\) if and only if a literal of \(x\) belongs to \(c\), then such a graph would be non-planar. The NAE3SAT problem on planar instances is, in fact, polynomial \cite{28}. Hence, \(G\) has to be assumed non-planar as well.

The main ingredient of the reduction is the fork gadget (Fig. 2), for which the following lemma holds (the proof is in Appendix A.1).

**Lemma 1.** Let \(G\) be an undirected graph containing a fork gadget \(F\) that does not contain the vertices \(s\) or \(t\). In any non-transitive \(st\)-orientation of \(G\), the

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fork_gadget.png}
\caption{Fig. 2. (a) The fork gadget. (b)-(c) The two possible orientations of the fork gadget in a non-transitive \(st\)-orientation of the whole graph.}
\end{figure}
edges \( e_9 \) and \( e_{10} \) of \( F \) are oriented either both exiting \( F \) or both entering \( F \). They are oriented exiting \( F \) if and only if edge \( e_1 \) is oriented entering \( F \).

For each Boolean variable \( x \) of \( \phi \) we construct a variable gadget \( V_x \) by suitably combining two fork gadgets, denoted \( F_x \) and \( F_{\overline{x}} \), as follows (see Fig. 3). We introduce two paths \( P_x \) and \( P_{\overline{x}} \) of length four from \( s \) to \( t \). The edge \( e_1 \) of \( F_x \) (of \( F_{\overline{x}} \), respectively) is attached to the middle vertex of path \( P_x \) (of path \( P_{\overline{x}} \), respectively). Edge \( e_{10} \) of \( F_x \) is identified with edge \( e_9 \) of \( F_{\overline{x}} \). The two edges \( e_9 \) of \( F_x \) and \( e_{10} \) of \( F_{\overline{x}} \) are denoted \( x \) and \( \overline{x} \), respectively. We have the following lemma (see Appendix A.1 for the proof).

**Lemma 2.** Let \( G \) be an undirected graph containing a variable gadget \( V_x \). In any non-transitive st-orientation of \( G \) the two edges of \( V_x \) denoted \( x \) and \( \overline{x} \) are one entering and one exiting \( V_x \) or vice versa.

By virtue of Lemma 2 we associate the \textbf{true} value of variable \( x \) with the orientation of \( V_x \) where edge \( x \) is oriented exiting and edge \( \overline{x} \) is oriented entering \( V_x \) (see Fig. 3(a)). We call such an orientation the \textbf{true orientation} of \( V_x \). Analogously, we associate the \textbf{false} value of variable \( x \) with the orientation of \( V_x \) where edge \( x \) is oriented entering and edge \( \overline{x} \) is oriented exiting \( V_x \) (see Fig. 3(b)). Observe that edge \( x \) (edge \( \overline{x} \), respectively) is oriented exiting \( V_x \) when the literal \( x \) (the literal \( \overline{x} \), respectively) is \textbf{true}. Otherwise edge \( x \) (edge \( \overline{x} \), respectively) is oriented entering \( V_x \).

The \textit{split gadget} \( S_k \) is composed of a chain of \( k-1 \) fork gadgets \( F_1, F_2, \ldots F_{k-1} \), where, for \( i = 1, 2, \ldots, k-2 \), the edge \( e_9 \) of \( F_i \) is identified with the edge \( e_1 \) of \( F_{i+1} \). We call \textit{input edge of} \( S_k \) the edge denoted \( e_1 \) of \( F_1 \). Also, we call \textit{output
Fig. 4. The split gadget $S_k$.

Fig. 5. The clause gadget $C_c$ for clause $c = (x_1 \lor x_2 \lor \overline{x}_3)$. The configurations of the three variable gadgets correspond to the truth values $x_1 = \text{true}$, $x_2 = \text{false}$, and $x_3 = \text{true}$. The clause is satisfied because the first literal $x$ is $\text{true}$ and the second and third literals $x_2$ and $\overline{x}_3$ are $\text{false}$.

**Lemma 3.** Let $G$ be an undirected graph containing a split gadget $S_k$ that does not contain the vertices $s$ or $t$. In any non-transitive st-orientation of $G$, the $k$ output edges of $S_k$ are all oriented exiting $S_k$ if the input edge of $S_k$ is oriented entering $S_k$. Otherwise, if the input edge of $S_k$ is oriented exiting $S_k$, the output edges of $S_k$ are all oriented entering $S_k$.

If the directed literal $x$ (negated literal $\overline{x}$, respectively) occurs in $k$ clauses, we attach the edge denoted $x$ (denoted $\overline{x}$, respectively) of $V_x$ to a split gadget $S_x$, and use the $k$ output edges of $S_x$ to carry the truth value of $x$ (of $\overline{x}$, respectively) to the $k$ clauses. The **clause gadget** $C_c$ for a clause $c = (l_1 \lor l_2 \lor l_3)$ is simply a vertex $v_c$ that is incident to three edges encoding the truth values of the three literals $l_1$, $l_2$, and $l_3$ (see Fig. 5). We prove the following.

**Theorem 1.** NTO is NP-complete.
Sketch of proof: The reduction from an instance $\varphi$ of NAE3SAT to an instance $I_\varphi$ described above is performed in time linear in the size of $\varphi$. Also, $I_\varphi$ is positive if and only if $\varphi$ is positive. Indeed, in any non-transitive $st$-orientation of $G$ each vertex $v_c$ of a clause gadget $C_c$ has at least one incoming and one outgoing edge, as well as in any truth assignment that satisfies $\varphi$ each clause $c$ has at least one true and one false literal. Finally, NTO is trivially in NP, as one can non-deterministically explore all possible orientations of the graph. □

The analogous problem where the source and the target vertices of $G$ are not prescribed but can be freely chosen is also NP-complete (see Appendix A.1).

3 ILP Model for Planar Graphs

Let $G$ be a planar graph with two prescribed vertices $s$ and $t$, such that $G \cup (s,t)$ is biconnected and such that $G$ admits a planar embedding with $s$ and $t$ on the external face. In this section we describe how to compute an $st$-orientation of $G$ with the minimum number of transitive edges by solving an ILP model.

Suppose that $G'$ is the plane $st$-graph resulting from a planar $st$-orientation of $G$, along with a planar embedding where $s$ and $t$ are on the external face. It is well known (see, e.g., [9]) that for each vertex $v \neq s, t$ in $G'$, all incoming edges of $v$ (as well as all outgoing edges of $v$) appear consecutively around $v$. Thus, the circular list of edges incident to $v$ can be partitioned into two linear lists, one containing the incoming edges of $v$ and the other containing the outgoing edges of $v$. Also, the boundary of each internal face $f$ of $G'$ consists of two edge-disjoint directed paths, called the left path and the right path of $f$, sharing the same end-vertices (i.e., the same source and the same destination). It can be easily verified that an edge $e$ of $G'$ is transitive if and only if it coincides with either the left path or the right path of some face of $G'$ (see also Claim 2 in [22]). Note that, since the transitivity of $e$ does not depend on the specific planar embedding of $G'$, the aforementioned property for $e$ holds for every planar embedding of $G'$. Due to this observation, in order to compute a planar $st$-orientation of $G$ with the minimum number of transitive edges, we can focus on any arbitrarily chosen planar embedding of $G$ with $s$ and $t$ on the external face.

Let $e_1$ and $e_2$ be two consecutive edges encountered moving clockwise along the boundary of a face $f$, and let $v$ be the vertex of $f$ shared by $e_1$ and $e_2$. The triple $(e_1, v, e_2)$ is an angle of $G$ at $v$ in $f$. Denote by $\deg(f)$ the number of angles in $f$ and by $\deg(v)$ the number of angles at $v$. As it was proved in [15], all planar $st$-orientations of the plane graph $G$ can be characterized in terms of labelings of the angles of $G$. Namely, each planar $st$-orientation of $G$ has a one-to-one correspondence with an angle labeling, called an $st$-labeling of $G$, that satisfies the following properties:

(L1) Each angle is labeled either S (small) or F (flat), except the angles at $s$ and at $t$ in the external face, which are not labeled;

(L2) Each internal face $f$ has 2 angles labeled S and $\deg(f) - 2$ angles labeled F;
Fig. 6. (a) An st-labeling of a plane graph $G$ with prescribed nodes $s$ and $t$. (b) The corresponding st-orientation of $G$.

(L3) For each vertex $v \neq s, t$ there are $\deg(v) - 2$ angles at $v$ labeled S and 2 angles at $v$ labeled F;

(L4) All angles at $s$ and $t$ in their incident internal faces are labeled S.

Given an st-labeling of $G$, the corresponding st-orientation of $G$ is such that for each vertex $v \neq s, t$, the two F angles at $v$ separate the list of incoming edges of $v$ to the list of outgoing edges of $v$, while the two S angles in a face $f$ separate the left and the right path of $f$. See Fig. 6 for an illustration. The st-orientation can be constructed from the st-labeling in linear time by a breadth-first-search of $G$ that starts from $s$, makes all edges of $s$ outgoing, and progressively orients the remaining edges of $G$ according to the angle labels.

Thanks to the characterization above, an edge $e = (u, v)$ of the st-graph resulting from an st-orientation is transitive if and only if in the corresponding st-labeling the angle at $u$ and the angle at $v$ in one of the two faces incident to $e$ (possibly in both faces) are labeled S. Based on this, we present an ILP model that describes the possible st-labelings of $G$ (for any arbitrary planar embedding of $G$ with $s$ and $t$ on the external face) and that minimizes the number of transitive edges. The model aims to assign angle labels that satisfy Properties (L1)–(L4) and counts pairs of consecutive S labels that occur in the circular list of angles in an internal face; additional constraints are needed to avoid that a transitive edge is counted twice when it coincides with both the left and the right path of its two incident faces. The model, which uses a number of variables and constraints that is linear in the size of $G$, is as follows.

Sets. Denote by $V$, $E$, and $F$ the sets of vertices, edges, and faces of $G$, respectively. Also let $F_{\text{int}} \subseteq F$ be the set of internal faces of $G$. For each face $f \in F$, let $V(f)$ and $E(f)$ be the set of vertices and the set of edges incident to
\( f \), respectively. For each vertex \( v \in V \), let \( F(v) \) be the set of faces incident to \( v \) and let \( F_{\text{int}}(v) \) be the set of internal faces incident to \( v \). For each edge \( e \in E \), let \( F(e) \) be the set consisting of the two faces incident to \( e \).

**Variables.** We define a binary variable \( x_{vf} \) for each vertex \( v \in V \setminus \{s, t\} \) and for each face \( f \in F(v) \). Also, we define the binary variables \( x_{sf} \) (resp. \( x_{tf} \)) for each face \( f \in F_{\text{int}}(s) \) (resp. \( f \in F_{\text{int}}(t) \)). If \( x_{vf} = 1 \) (resp. \( x_{vf} = 0 \)) we assign an S label (resp. an F label) to the angle at \( v \) in \( f \).

For each internal face \( f \in F_{\text{int}} \) and for each edge \( (u, v) \in E(f) \), we define a binary variable \( y_{uvf} \). An assignment \( y_{uvf} = 1 \) indicates that both the angles at \( u \) and at \( v \) in \( f \) are labeled S, that is, \( x_{uf} = 1 \) and \( x_{vf} = 1 \). As a consequence, if \( y_{uvf} = 1 \) edge \((u, v)\) is transitive. Note however that the sum of all \( y_{uvf} \) does not always correspond to the number of transitive edges; indeed, if \( f \) and \( g \) are the two internal faces incident to edge \((u, v)\), it may happen that both \( y_{uvf} \) and \( y_{uvg} \) are set to one, thus counting \((u, v)\) as transitive twice. To count the number of transitive edges without repetitions, we introduce another binary variable \( z_{uv} \), for each edge \((u, v)\) \( \in E \), such that \( z_{uv} = 1 \) if and only if \((u, v)\) is transitive.

**Objective function and constraints.** The objective function and the set of constraints are described by the formulas (1)–(8). The objective is to minimize the total number of transitive edges, i.e., the sum of the variables \( z_{uv} \). Constraints 2 and 3 guarantee Properties (L2) and (L3) of the \( st \)-labeling, respectively, while Constraints 4 and 5 guarantee Property (L4). Constraints 6 relate the values of the variables \( y_{uvf} \) to the values of \( x_{uf} \) and \( x_{vf} \). Namely, they guarantee that \( y_{uvf} = 1 \) if and only if both \( x_{uf} \) and \( x_{vf} \) are set to 1. Constraints 7 relate the values of the variables \( z_{uv} \) to those of the variables \( y_{uvf} \); they guarantee that an edge \((u, v)\) is counted as transitive (i.e., \( z_{uv} = 1 \)) if and only if in at least one of the two faces \( f \) incident to \((u, v)\) both the angle at \( u \) and the angle at \( v \) are labeled S. Finally, we explicitly require that \( x_{uv} \) and \( y_{uv} \) are binary variables, while we only require that each \( z_{uv} \) is a non-negative integer; this helps to speed-up the solver and, along with the objective function, is enough to guarantee that each \( z_{uv} \) takes value 0 or 1.

\[
\min \sum_{(u, v) \in E} z_{uv} \tag{1}
\]

\[
\sum_{v \in V(f)} x_{vf} = 2 \quad \forall f \in F_{\text{int}} \tag{2}
\]

\[
\sum_{f \in F(v)} x_{vf} = \deg(v) - 2 \quad \forall v \in V \setminus \{s, t\} \tag{3}
\]

\[
x_{sf} = 1 \quad \forall f \in F_{\text{int}} \cap F(s) \tag{4}
\]

\[
x_{tf} = 1 \quad \forall f \in F_{\text{int}} \cap F(t) \tag{5}
\]

\[
x_{uf} + x_{vf} \leq y_{uvf} + 1 \quad \forall f \in F_{\text{int}} \quad \forall (u, v) \in E(f) \tag{6}
\]

\[
z_{uv} \geq y_{uvf} \quad \forall e = (u, v) \in E \quad \forall f \in F(e) \tag{7}
\]

\[
x_{vf} \in \{0, 1\} \quad y_{uvf} \in \{0, 1\} \quad z_{uv} \in \mathbb{N} \tag{8}
\]
4 Experimental Analysis

We evaluated the ILP model with the solver IBM ILOG CPLEX 20.1.0.0 (using the default setting), running on a laptop with Microsoft Windows 11 v.10.0.22000 OS, Intel Core i7-8750H 2.20GHz CPU, and 16GB RAM.

Instances. The experiments have been executed on a large benchmark of instances, each instance consisting of a plane biconnected graph and two vertices $s$ and $t$ on the external face. These graphs are randomly generated with the same approach used in previous experiments in graph drawing (see, e.g., [3]). Namely, for a given integer $n > 0$, we generate a plane graph with $n$ vertices starting from a triangle and executing a sequence of steps, each step preserving biconnectivity and planarity. At each step the procedure randomly performs one of the two following operations: (i) an Insert-Edge operation, which splits a face by adding a new edge, or (ii) an Insert-Vertex operation, which subdivides an existing edge with a new vertex. The Insert-Vertex operation is performed with a prescribed probability $p_{iv}$ (which is a parameter of the generation process), while the Insert-Edge operation is performed with probability $1 - p_{iv}$. For each operation, the elements (faces, vertices, or edges) involved are randomly selected with uniform probability distribution. To avoid multiple edges, if an Insert-Edge operation selects two end-vertices that are already connected by an edge, we discard the selection and repeat the step. Once the plane graph is generated, we randomly select two vertices $s$ and $t$ on its external face, again with uniform probability distribution. We generated a sample of 10 instances for each pair $(n, p_{iv})$, with $n \in \{10, 20, \ldots, 90, 100, 200, \ldots, 900, 1000\}$ and $p_{iv} \in \{0.2, 0.4, 0.5, 0.6, 0.8\}$, for a total of 950 graphs. Note that, higher values of $p_{iv}$ lead to sparser graphs.

Table 1 in the appendix reports for each sample the average, the minimum, and the maximum density (number of edges divided by the number of vertices) of the graphs in that sample, together with the standard deviation. On average, for $p_{iv} = 0.8$ we have graphs with density of 1.23 (close to the density of a tree), for $p_{iv} = 0.5$ we have graphs with density of 1.76, and for $p_{iv} = 0.2$ we have graphs with density 2.53 (close to the density of maximal planar graphs).

Experimental Goals. We have three main experimental goals: (G1) Evaluate the efficiency of our approach, i.e., the running time required by our ILP model; (G2) Evaluate the percentage of transitive edges in the solutions of the ILP model and how many transitive edges are saved w.r.t. applying a classical linear-time algorithm that computes an unconstrained $st$-orientation of the graph [20]; (G3) Evaluate the impact of minimizing the number of transitive edges on the area (i.e. the area of the minimum bounding box) of polyline drawings constructed with algorithms that compute an $st$-orientation as a preliminary step.

About (G1), we refer to the algorithm that solves the ILP model as OptST. About (G2) and (G3) we used implementations available in the GDToolkit library [8] for the following algorithms: (a) A linear-time algorithm that computes an unconstrained $st$-orientation of the graph based on the classical $st$-numbering algorithm by Even and Tarjan [20]. We refer to this algorithm as HeurST. (b) A linear-time algorithm that first computes a visibility representation of
an undirected planar graph based on a given \( st \)-orientation of the graph, and then computes from this representation a planar polyline drawing [10]. We call \textsc{DrawHeurST} and \textsc{DrawOptST} the applications of this drawing algorithm to the \( st \)-graphs obtained by \textsc{HeurST} and of \textsc{OptST}, respectively.

**Experimental Results.** About (G1), Fig. 7 reports the running time (in seconds) of \textsc{OptST}, i.e., the time needed by CPLEX to solve our ILP model. To make the charts more readable we split the results into two sets, one for the instances with number of vertices up to 90 and the other for the larger instances. \textsc{OptST} is rather fast: 75% of the instances with up to 90 vertices is solved in less than one second and all these instances are solved in less than five seconds. For the larger instances (with up to 1000 vertices), 75% of the instances are solved in less than 10 seconds and all instances are solved in less than 25 seconds. These results clearly indicate that our ILP model can be successfully used in several application contexts that manage graphs with up to thousand vertices.

About (G2), Fig. 8 shows the reduction (in percentage) of the number of transitive edges in the solutions of \textsc{OptST} with respect to the solutions of \textsc{HeurST}. More precisely, Fig. 8(a) reports values averaged over all instances with the same number of vertices; Fig. 8(b), Fig. 8(c), and Fig. 8(d) report the same data, partitioning the instances by different values of \( p_v \), namely 0.8 (the sparsest instances), 0.4-0.6 (instances of medium density), and 0.2 (the densest instances). For each instance, denoted by \( tr_{\text{Opt}} \) and \( tr_{\text{Heur}} \) the number of transitive edges of the solutions computed by \textsc{OptST} and \textsc{HeurST}, respectively, the reduction percentage equals the value \( \left( \frac{tr_{\text{Heur}} - tr_{\text{Opt}}}{\max\{1, tr_{\text{Heur}}\}} \times 100 \right) \). Over all instances, the average reduction is about 35%; it grows above 60% on the larger graphs if we restrict to the sparsest instances (with improvements larger than 80% on some graphs), while it is below 30% for the densest instances, due to the presence of many 3-cycles, for which a transitive edge cannot be avoided.

About (G3), Fig. 9 shows the percentage of instances for which \textsc{DrawOptST} produces drawings that are better than those produced by \textsc{DrawHeurST} in terms of area requirement (the label “better” of the legend). It can be seen that \textsc{DrawOptST} computes more compact drawings for the majority of the instances. In particular, it is interesting to observe that this is most often the...
Fig. 8. Improvement (%) in the number of transitive edges.

case even for the densest instances (i.e., those for $p_{iv} = 0.2$), for which we have previously seen that the average reduction of transitive edges is less evident. For those instances for which DrawOptST computes more compact drawings than DrawHeurST, Fig. 10 reports the average percentage of improvement in terms of area requirement (i.e., the percentage of area reduction). The values are mostly between 30% and 50%. To complement this data, Fig. 11 reports the trend of the improvement (reduction) in terms of drawing area with respect to the reduction of the transitive edges (discretized in four intervals). For the instances with $p_{iv} = 0.8$ and $p_{iv} = 0.2$, the correlation between these two measures is quite evident. For the instances of medium density ($p_{iv} \in \{0.4, 0.5, 0.6\}$), the highest values of improvement in terms of area requirement are observed for reductions of transitive edges between 22% and 66%. Figures 13 and 14 in the appendix show drawings computed by DrawHeurST and DrawOptST for two of our instances.

5 Final Remarks and Open Problems

We addressed the problem of computing $st$-orientations with the minimum number of transitive edges. This problem has practical applications in graph drawing, as finding an $st$-orientation is at the heart of several graph drawing algorithms. Although $st$-orientations without transitive edges have been studied from a combinatorial perspective [22], there is a lack of practical algorithms, and the com-
Fig. 9. Instances for which DrawOptST produces drawings that are more compact than DrawHeurST (label “better”).

The complexity of deciding whether a graph can be oriented to become an st-graph without transitive edges seems not to have been previously addressed.

We proved that this problem is NP-hard in general and we described an ILP model for planar graphs based on characterizing planar st-graphs without transitive edges in terms of a constrained labeling of the vertex angles inside its faces. An extensive experimental analysis on a large set of instances shows that our model is fast in practice, taking few seconds for graphs of thousand vertices. It saves on average 35% of transitive edges w.r.t. a classical algorithm that computes an unconstrained st-orientation. We also showed that for classical layout algorithms that compute polyline drawings of planar graphs through an st-orientation, minimizing the number of transitive edges yields more compact drawings most of the time (see also Fig. 13 and Fig. 14 in the appendix).

We suggest two future research directions: (i) It remains open to establish the time complexity of the problem for planar graphs. Are there polynomial-time algorithms that compute st-orientations with the minimum number of transitive edges for all planar graphs or for specific subfamilies of planar graphs? (ii) One can extend the experimental analysis to real-world graphs and design fast heuristics, which can be compared to the optimal algorithm.
Fig. 10. Area improvement (%) of DrawOptST w.r.t. DrawHeurST, for the instances where DrawOptST is “better” (i.e., the “better” instances in Fig. 9).

Fig. 11. Correlation between the improvement (reduction) in terms of drawing area and in terms of transitive edges improvement.
References

1. Alzohairi, M., Rival, I.: Series-parallel planar ordered sets have pagewidth two. In: Graph Drawing. Lecture Notes in Computer Science, vol. 1190, pp. 11–24. Springer (1996)
2. Angelini, P., Cittadini, L., Didimo, W., Frati, F., Di Battista, G., Kaufmann, M., Symvonis, A.: On the perspectives opened by right angle crossing drawings. J. Graph Algorithms Appl. 15(1), 53–78 (2011)
3. Bertolazzi, P., Di Battista, G., Didimo, W.: Computing orthogonal drawings with the minimum number of bends. IEEE Trans. Computers 49(8), 826–840 (2000)
4. Binucci, C., Didimo, W.: Computing quasi-upward planar drawings of mixed graphs. Comput. J. 59(1), 133–150 (2016)
5. Binucci, C., Didimo, W., Patrignani, M.: Upward and quasi-upward planar testing of embedded mixed graphs. Theor. Comput. Sci. 526, 75–89 (2014)
6. Brandes, U.: Eager st-ordering. In: ESA. Lecture Notes in Computer Science, vol. 2461, pp. 247–256. Springer (2002)
7. Chimani, M., Gutwenger, C., Jünger, M., Klau, G.W., Klein, K., Mutzel, P.: The open graph drawing framework (OGDF). In: Handbook of Graph Drawing and Visualization, pp. 543–569. Chapman and Hall/CRC (2013)
8. Di Battista, G., Didimo, W.: Gdtoolkit. In: Handbook of Graph Drawing and Visualization, pp. 571–597. Chapman and Hall/CRC (2013)
9. Di Battista, G., Tamassia, R.: Algorithms for plane representations of acyclic digraphs. Theor. Comput. Sci. 61, 175–198 (1988)
10. Di Battista, G., Tamassia, R., Tollis, I.G.: Area requirement and symmetry display of planar upward drawings. Discret. Comput. Geom. 7, 381–401 (1992)
11. Di Battista, G., Tamassia, R., Tollis, I.G.: Book embeddability of series-parallel digraphs. Algorithmica 45(4), 531–547 (2006)
12. Dickerson, M., Eppstein, D., Goodrich, M.T., Meng, J.Y.: Confluent drawings: Visualizing non-planar diagrams in a planar way. J. Graph Algorithms Appl. 9(1), 31–52 (2005)
13. Didimo, W.: Upward graph drawing. In: Encyclopedia of Algorithms, pp. 2308–2312 (2016)
14. Eppstein, D., Simons, J.A.: Confluent Hasse diagrams. J. Graph Algorithms Appl. 17(7), 689–710 (2013)
15. Even, S., Tarjan, R.E.: Computing an st-numbering. Theor. Comput. Sci. 2(3), 339–344 (1976)
16. Even, S., Tarjan, R.E.: Corrigendum: Computing an st-numbering. TCS 2(1976):339-344. Theor. Comput. Sci. 4(1), 123 (1977)
17. Frati, F., Kaufmann, M., Pach, J., Tóth, C.D., Wood, D.R.: On the upward planarity of mixed plane graphs. J. Graph Algorithms Appl. 18(2), 253–279 (2014)
18. Fusy, É., Naramel, E., Schaeffer, G.: On the enumeration of plane bipolar posets and transversal structures. CoRR abs/2105.06955 (2021)
23. Healy, P., Nikolov, N.S.: Hierarchical drawing algorithms. In: Handbook of Graph Drawing and Visualization, pp. 409–453. Chapman and Hall/CRC (2013)
24. Jünger, M., Mutzel, P. (eds.): Graph Drawing Software. Springer (2004)
25. Kaufmann, M., Wagner, D. (eds.): Drawing Graphs, Methods and Models (the book grow out of a Dagstuhl Seminar, April 1999), Lecture Notes in Computer Science, vol. 2025. Springer (2001)
26. Lempel, A., Even, S., Cederbaum, I.: An algorithm for planarity testing of graphs. In: Theory of Graphs: Internat. Symposium (Rome 1966). pp. 215–232. Gordon and Breach, New York (1967)
27. McConnell, R.M., Spinrad, J.P.: Modular decomposition and transitive orientation. Discret. Math. 201(1-3), 189–241 (1999)
28. Moret, B.M.E.: Planar NAE3SAT is in P. SIGACT News 19(2), 51–54 (1988)
29. Papamanchou, C., Tollis, I.G.: Algorithms for computing a parameterized st-orientation. Theor. Comput. Sci. 408(2-3), 224–240 (2008)
30. Papamanchou, C., Tollis, I.G.: Applications of parameterized st-orientations. J. Graph Algorithms Appl. 14(2), 337–365 (2010)
31. Platt, C.: Planar lattices and planar graphs. Journal of Combinatorial Theory, Series B 21(1), 30–39 (1976)
32. Rosenstiehl, P., Tarjan, R.E.: Rectilinear planar layouts and bipolar orientations of planar graphs. Discret. Comput. Geom. 1, 343–353 (1986)
33. Schaefer, T.J.: The complexity of satisfiability problems. In: Proc. of the 10th Annual ACM Symposium on Theory of Computing, pp. 216–226 (1978)
34. Wiese, R., Eiglsperger, M., Kaufmann, M.: yFiles - visualization and automatic layout of graphs. In: Graph Drawing Software, pp. 173–191. Springer (2004)
A Appendix

A.1 Additional Material for Section 2

Observation 1. Let \((v_1, v_2, \ldots, v_k)\) be a path of \(G\) such that its internal vertices \(v_2, v_3, \ldots, v_{k-1}\) have degree 2 in \(G\) and are different from \(s\) and \(t\). In any non-transitive \(st\)-orientation of \(G\) the edges \((v_i, v_{i+1})\), with \(i = 1, \ldots, k - 1\), are all directed from \(v_i\) to \(v_{i+1}\) or they are all directed from \(v_{i+1}\) to \(v_i\).

Proof. The statement can be easily proved by observing that if two edges of the path have an inconsistent orientation (as in Fig. 12(c)) then the path would contain an internal vertex that is a source or a sink different from \(s\) and \(t\), contradicting the hypothesis that the orientation is an \(st\)-orientation.

![Fig. 12. (a) A path of \(G\) with all internal vertices of degree two. (b) A consistent orientation of the path. (c) An inconsistent orientation of the path generates sinks or sources. (d) A directed path of \(G\) and a chord.](image)

Observation 2. Let \((v_1, v_2, \ldots, v_k)\) be a path of \(G\) and let \((v_1, v_k)\) be an edge of \(G\). In any non-transitive \(st\)-orientation of \(G\) the edges \((v_i, v_{i+1})\), with \(i = 1, \ldots, k - 1\), cannot be all directed from \(v_i\) to \(v_{i+1}\).

Proof. Suppose for a contradiction that there exists a non-transitive \(st\)-orientation of \(G\) such that each edge \((v_i, v_{i+1})\), with \(i = 1, \ldots, k - 1\), is directed from \(v_i\) to \(v_{i+1}\) (refer to Fig. 12(d)). If edge \((v_1, v_k)\) was also directed from \(v_1\) to \(v_k\) it would be a transitive edge, contradicting the hypothesis that the orientation is non-transitive. Otherwise, if \((v_1, v_k)\) was directed from \(v_k\) to \(v_1\) it would form a directed cycle, contradicting the hypothesis that the orientation is an \(st\)-orientation.

Proof of Lemma 1

Lemma 1. Let \(G\) be an undirected graph containing a fork gadget \(F\) that does not contain the vertices \(s\) or \(t\). In any non-transitive \(st\)-orientation of \(G\), the edges \(e_9\) and \(e_{10}\) of \(F\) are oriented either both exiting \(F\) or both entering \(F\). They are oriented exiting \(F\) if and only if edge \(e_1\) is oriented entering \(F\).
Proof. Suppose edge $e_1$ is oriented entering $F$ (refer to Fig. 2(b)). One between $e_9$ or $e_{10}$ must be oriented exiting $F$, otherwise $F$ contains a sink contradicting the fact that we have an $st$-orientation of $G$. Since gadget $F$ is symmetric, we may assume without loss of generality that edge $e_9$ is oriented exiting $F$. Therefore, there must be at least one directed path from $e_1$ to $e_9$ traversing $F$. There are three possible such directed paths: (1) path $(e_1, e_4, e_8, e_7, e_6, e_9)$; (2) path $(e_1, e_3, e_6, e_9)$; and (3) path $(e_1, e_2, e_5, e_9)$. Suppose Case (1) applies, i.e., $(e_1, e_4, e_8, e_7, e_6, e_9)$ is a directed path. We have a contradiction because of Observation 2 applied to the directed path $(e_4, e_8, e_7)$ and the chord $e_3$. Suppose Case (2) applies, i.e., $(e_1, e_3, e_6, e_9)$ is a directed path. Note that by Observation 1 the edges $e_2$ and $e_9$ must be both directed in the same direction. If they were directed towards $v$, then we would have a directed cycle $(e_4, e_6, e_5, e_2)$. Hence, $(e_2, e_5)$ are directed away from $v$ and, since $(e_1, e_2, e_5, e_9)$ is also a directed path, Case (2) implies Case (3). Conversely, suppose Case (3) applies, i.e., $(e_1, e_2, e_5, e_9)$ is a directed path. Edge $e_6$ must be directed towards $w$. In fact, if $e_6$ was directed away from $w$ we would have a contradiction by Observation 2 applied to the directed path $(e_2, e_5, e_6)$ and the chord $e_3$. Also, edge $e_3$ must be directed away from $v$. In fact, if $e_3$ was directed towards $v$ edge $e_6$ would be a transitive edge with respect to the directed path $(e_3, e_2, e_5)$. It follows that $(e_1, e_3, e_6, e_9)$ would also be a directed path and Case (3) implies Case (2). Therefore, we have to assume that Case (2) and Case (3) both apply. Note that by Observation 1 the edges $e_4$ and $e_8$ must be both directed in the same direction. If the path $(e_8, e_4)$ was oriented entering $z$ and entering $v$ then we would have a contradiction because of Observation 2 applied to the directed path $(e_8, e_4, e_3)$ and the chord $e_7$. It follows that the path $(e_4, e_8)$ is oriented exiting $v$ and entering $z$. Now, edge $e_7$ must be oriented exiting $z$, otherwise $e_3$ would be a transitive edge with respect to the path $(e_4, e_8, e_7)$. Finally, edge $e_{10}$ must be oriented exiting $z$, otherwise $z$ would be a sink. In conclusion, if $e_1$ is oriented entering $F$, then $e_9$ and $e_{10}$ must be oriented exiting $F$.

With analogous and symmetric arguments it can be proved that if $e_1$ is oriented exiting $F$ (refer to Fig. 2(c)), then $e_9$ and $e_{10}$ must be oriented entering $F$. Since $e_1$ must be oriented in one way or the other, the only two possible orientations of $F$ are those depicted in Figs. 2(b) and 2(c) and the statement follows.

Proof of Lemma 2

Lemma 2. Let $G$ be an undirected graph containing a variable gadget $V_x$. In any non-transitive $st$-orientation of $G$ the two edges of $V_x$ denoted $x$ and $\bar{x}$ are one entering and one exiting $V_x$ or vice versa.

Proof. Suppose edge $e_1$ of $F_x$ is oriented entering $F_x$ (see Fig. 3(a)). By Lemma 1 edge $x$ is oriented exiting $F_{\bar{x}}$ and, hence, exiting $V_{\bar{x}}$. Also edge $e_9$ of $F_x$, which coincides with $e_{10}$ of $F_{\bar{x}}$, is oriented exiting $F_x$ and entering $F_{\bar{x}}$. Now, always by Lemma 1, edge $e_1$ of $F_{\bar{x}}$ is oriented exiting $F_{\bar{x}}$ and edge $e_9$ of $F_{\bar{x}}$, which coincides with edge $\bar{x}$ of $V_x$, is oriented entering $F_{\bar{x}}$ and, hence, entering $V_x$. 


Suppose now that edge $e_1$ of $F_x$ is oriented exiting $F_x$ (see Fig. 3(b)). By Lemma 1 edge $x$ is oriented entering $F_x$ and, hence, entering $V_x$. Also edge $e_9$ of $F_x$, which coincides with $e_10$ of $F_x$, is oriented entering $F_x$ and exiting $F_x$. Now, always by Lemma 1, edge $e_1$ of $F_x$ is oriented entering $F_x$ and edge $e_{90}$ of $F_x$, which coincides with edge $\tau$ of $V_x$, is oriented exiting $F_x$ and, hence, exiting $V_x$.

Finally, observe that, even if a directed path was added outside $V_x$ from edge $x$ to edge $\tau$ or vice versa, no directed cycle traverses $V_x$. In fact, all directed paths exiting $V_x$ originate from $s$ and all directed paths entering $V_x$ go to $t$.

**Proof of Theorem 1**

Theorem 1. NTO is NP-complete.

Proof. The reduction from an instance $\varphi$ of NAE3SAT to an instance $I_\varphi$ previously described is performed in time linear in the size of $\varphi$.

Suppose $I_\varphi = (G, s, t)$ is a positive instance of NTO and consider any non-transitive $st$-orientation of $G_\varphi$. Consider a clause $c$ of $\varphi$ and the corresponding vertex $v_c$ in $G$. Since vertex $v_c$ is not a sink nor a source it must have at least one entering edge $e_{in}$ and at least one exiting edge $e_{out}$. Consider first edge $e_{in}$ and assume it corresponds to a directed literal $x_i$ of $c$ (to a negated literal $\overline{x}_i$ of $c$, respectively). By construction, edge $e_{in}$ comes from the edge $x_i$ (edge $\overline{x}_i$, respectively) of variable gadget $V_{x_i}$ or from an intermediate split gadget $S_{x_i}$ ($S_{\overline{x}_i}$, respectively) that has edge $x_i$ (edge $\overline{x}_i$, respectively) as input edge. Therefore, by Lemmas 2 and 3 edge $x$ (edge $\overline{x}_i$, respectively) of $V_{x_i}$ is oriented exiting $V_{x_i}$, which corresponds to a true literal of $c$. Now consider edge $e_{out}$ and assume it corresponds to a directed literal $x_j$ of $c$ (to a negated literal $\overline{x}_j$ of $c$, respectively). With analogous arguments as above you conclude that edge $x_j$ (edge $\overline{x}_j$, respectively) of $V_{x_j}$ is oriented entering $V_{x_j}$, which corresponds to a false literal of $c$. Therefore, each clause $c$ has both a true and a false literal and the NAE3SAT instance $\varphi$ is a yes instance.

Conversely, suppose that instance $\varphi$ is a yes instance of NAE3SAT. Consider a truth assignment to the variables in $X$ that satisfies $\varphi$. Orient the edges of each variable gadget $V_x$ as depicted in Fig. 3(a) or Fig. 3(b) depending on whether variable $x$ is set to true or false in the truth assignment, respectively. Orient each split gadget according to its input edge. Since the truth assignment is such that every clause has a true literal and a false literal, the corresponding clause gadget $C_c$ will have at least one incoming edge and one outgoing edge. Therefore the obtained orientation is a non-transitive $st$-orientation of $G$. Regarding acyclicity, observe that variable gadgets and clause gadgets whose edges are oriented as depicted in Fig. 3 and Fig. 5, respectively, are acyclic. Also, a split gadget whose output edges are oriented all exiting or all entering the gadget is acyclic. Since all the directed paths that enter a variable gadget $V_{x_i}$ terminate at $t$ without exiting $V_{x_i}$, and all the directed paths that leave $V_{x_i}$ come from $s$ without entering $V_{x_i}$, there cannot be a directed cycle involving a variable gadget $V_{x_i}$. It remains to show that there are no directed cycles involving split gadgets and clause gadgets. However, by Lemma 3 no directed path may enter a split
gadget from a clause gadget and exit the split gadget towards a second clause gadget. Hence, directed cycles involving clause gadgets and split gadgets alone cannot exist.

Finally, NTO is trivially in NP, as one can non-deterministically explore all possible orientations of the graph.

**Complexity of NTO where s and t can be freely chosen.** Observe that the variant of the NTO problem where the source and the target vertices of $G$ are not prescribed but can be freely chosen is also NP-hard. Problem NTO, in fact, can be easily reduced to it. Consider an instance $⟨G^*, s^*, t^*⟩$ of NTO. Add two vertices $s^+$ and $t^+$ to $G^*$ and connect them to $s^*$ and to $t^*$, respectively. Call $G^+$ the obtained graph. Since $s^+$ and $t^+$ have degree one in $G^+$, in any non-transitive $st$-orientation of $G^+$ they can only be sources or sinks, where if one of them is the source the other one is the sink. Hence, given any non-transitive $st$-orientation of $G^+$ you can immediately find a non-transitive $s^*t^*$-orientation of $G^*$, possibly by reversing all edge orientations if $t^+$ is the source and $s^+$ is the sink. Conversely, given a non-transitive $s^*t^*$-orientation of $G^*$ you easily find an $st$-orientation of $G$ orienting the edge $(s^+, s^*)$ from $s^+$ to $s^*$ and the edge $(t^*, t^+)$ from $t^*$ to $t^+$. Therefore, the addition of edges $(s^+, s^*)$ and $(t^+, t^*)$ is a polynomial-time reduction from problem NTO with prescribed source and target to the variant of the NTO problem where these vertices can be freely choosen, proving the hardness of the latter problem. Since this variant of NTO is also trivially in NP it is NP-complete.

**A.2 Additional Material for Section 4**
Fig. 13. Two polyline drawings of the same plane graph with 100 vertices and $p_{iv} = 0.8$ computed by (a) DrawHeurST and (b) DrawOptST. Transitive edges are colored red.
Fig. 14. Two polyline drawings of the same plane graph with 100 vertices and $p_{iv} = 0.5$ computed by (a) DRAWHEURST and (b) DRAWOPTST. Transitive edges are colored red.
| \( n \) | AVGMIN | MAX | SD | AVGMIN | MAX | SD | AVGMIN | MAX | SD |
|-----|-------|-----|----|-------|-----|----|-------|-----|----|
| 10  | 1.16  | 1.40 | 0.11 | 1.33  | 1.10 | 0.11 | 1.50  | 1.20 | 0.22 |
| 20  | 1.19  | 1.05 | 0.09 | 1.54  | 3.30 | 0.25 | 1.65  | 3.05 | 0.28 |
| 30  | 1.23  | 1.07 | 0.10 | 1.68  | 4.07 | 0.18 | 1.83  | 4.05 | 0.20 |
| 40  | 1.27  | 1.10 | 0.06 | 1.83  | 4.08 | 0.16 | 1.98  | 4.09 | 0.14 |
| 50  | 1.31  | 1.14 | 0.09 | 2.03  | 4.10 | 0.12 | 2.19  | 4.35 | 0.10 |
| 60  | 1.35  | 1.19 | 0.06 | 2.21  | 4.16 | 0.08 | 2.32  | 4.40 | 0.06 |
| 70  | 1.39  | 1.24 | 0.05 | 2.39  | 4.28 | 0.06 | 2.48  | 4.45 | 0.04 |
| 80  | 1.43  | 1.29 | 0.04 | 2.57  | 4.36 | 0.04 | 2.65  | 4.40 | 0.03 |
| 90  | 1.47  | 1.34 | 0.03 | 2.75  | 4.44 | 0.03 | 2.83  | 4.40 | 0.02 |
| 100 | 1.51  | 1.39 | 0.01 | 2.93  | 4.52 | 0.01 | 3.01  | 4.46 | 0.01 |

Table 1. Density of the different instances of our graph benchmark.