Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 27, no. 8, pp. 625–650 (2023)
DOI: 10.7155/jgaa.00638

st-Orientations with Few Transitive Edges

Carla Binucci [1] *Walter Didimo* [1] *Maurizio Patrignani* [2]

[1]Department of Engineering, University of Perugia, Perugia, Italy
[2]Department of Engineering, Roma Tre University, Rome, Italy

Submitted: November 2022	Reviewed: January 2023	Revised: March 2023
Accepted: April 2023	Final: April 2023	Published: November 2023

Article type: Regular paper

Communicated by: P. Angelini and R. von Hanxleden

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 (Integer Linear Programming) model that is fast in practice, namely it takes on average less than 1 second for graphs with up to 100 vertices, and about 10 seconds for larger instances with up to 1000 vertices. We experimentally show that optimum solutions significantly reduce (35% on average) 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 (with an improvement between 30% and 50% for most of the instances).

1 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

Special Issue on the 30th Int. Symposium on Graph Drawing and Network Visualization, GD 2022

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 RICBA22CB: Modelli, algoritmi e sistemi per la visualizzazione e l'analisi di grafi e reti.

E-mail addresses: carla.binucci@unipg.it (Carla Binucci) walter.didimo@unipg.it (Walter Didimo) maurizio.patrignani@uniroma3.it (Maurizio Patrignani)

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 [18] and right-angle-crossing drawings [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 are curves monotonically increasing in the same direction [4, 5, 16, 19, 23, 25].

Specific types of acyclic orientations that are central to many graph algorithms and applications are the so called *st-orientations*, also known as *bipolar orientations* [41], 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 admits an st-orientation if and only if G, with the addition of the edge (s, t) if not already present, is biconnected (i.e., the graph cannot be disconnected by removing a single vertex). The digraph resulting from an st-orientation is also called an *st-graph*. An st-orientation can be computed in linear time by first computing in linear time an st-numbering (or st-ordering) of the vertices of G [21], and then by orienting each edge from the end-vertex with smaller number to the end-vertex with larger number. A different algorithm that directly computes an st-orientation (and which uses it to compute an st-ordering) is given in [7]. In particular, if G is planar, a *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 [28], 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 [11, 27]). Planar st-orientations and related graph layout algorithms are at the heart of several graph drawing libraries and software (see, e.g., [9, 10, 26, 44]). Algorithms that compute st-orientations with specific characteristics (e.g., bounds on the length of the longest path) are also proposed and experimented in the context of visibility and orthogonal drawings [36, 37].

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* [11, 20], *bipolar posets* [24], or *Hasse diagrams of lattices* [12, 38]. 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 [13]; 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, in which each face consists of two edge-disjoint directed paths, called left and right paths, sharing their two end-vertices. 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

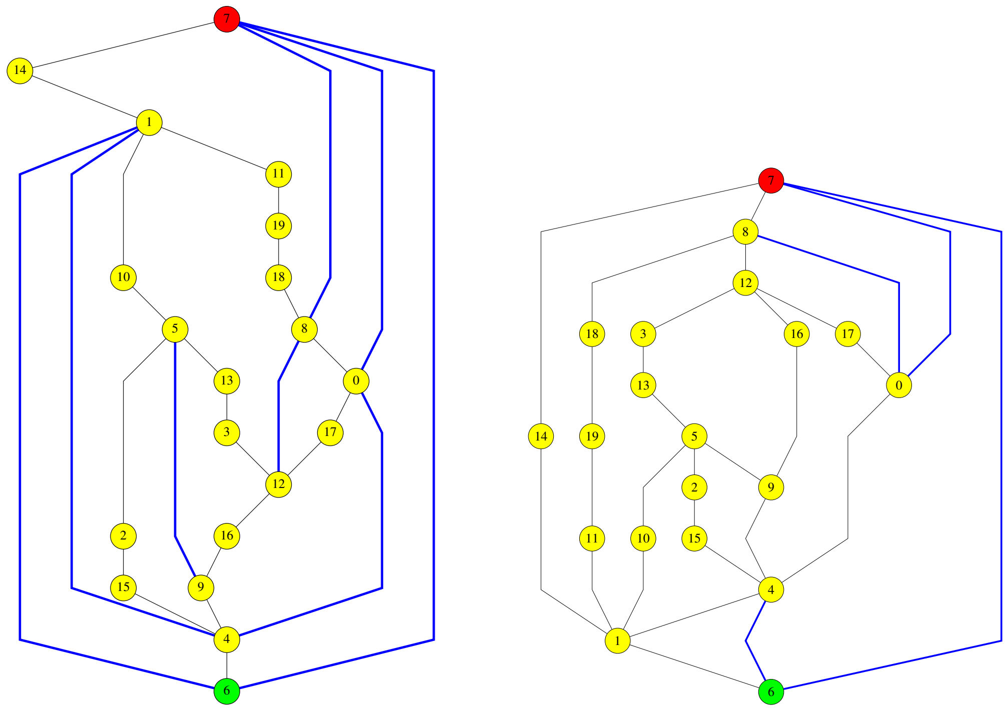


Figure 1: Two polyline drawings of the same plane graph, computed using two different *st*-orientations, with $s = 6$ (the green, bottommost vertex) and $t = 7$ (the red, topmost vertex); transitive edges are colored blue and thicker. All edges are drawn monotone in the upward direction. (a) An unconstrained *st*-orientation with 8 transitive edges, computed through an *st*-numbering; (b) An *st*-orientation with the minimum number (four) of transitive edges; the resulting drawing is more compact, it reduces the area by about 15%

.

result in long curves; avoiding them produces faces where the lengths of 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 [20]. Confluent drawings exploit edge bundling to create "planar" layouts of non-planar graphs, without introducing ambiguity [15]. 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 [14].

Contribution. The contribution of our paper is twofold:

- From a theoretical perspective, we prove that deciding whether a graph admits an *st*-orientation without transitive edges is NP-complete (Section 2). On the other hand, deciding whether an undirected graph has an orientation such that the resulting digraph coincides with its own transitive closure is linear-time solvable [30];

- 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 (Section 3). 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. The results of an extensive experimental analysis (Section 4) show that the ILP model works very fast in practice; popular solvers such as CPLEX can find a solution in about 10 seconds for instances with up to 1000 vertices. The number of transitive edges in the *st*-orientations computed by our model is on average 35% smaller than the one in the *st*-orientations computed with classical unconstrained algorithms; for some instances the improvement is greater than 80%. 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, with an improvement ranging from 30% to 50% for most of the instances.

2 NP-Completeness of the General Problem

The complexity of the problem of orienting the edges of an undirected graph so that the resulting digraph has no transitive-edge and no directed cycle has a long research history. This problem was first posed in 1962 by Ore [35] who asked to recognize the undirected graphs that can be oriented as the diagram of an ordered set. This is an equivalent formulation of the problem above, sometimes called the *cover graph recognition* problem. In 1987 Nešetřil and Rödl claimed to have proven that the cover graph recognition problem is NP-complete [33]. Unfortunately, in 1991 a flaw in their proof was discovered [43], forcing the authors to amend the issue [34]. In doing so, they relied on a result by Lund and Yannakakis [29] about the hardness of approximating the chromatic number of a graph. As the resulting proof was thought to be very complex, Brightwell came up with an alternative proof [8] which was much more simple, being a direct reduction from NAE3SAT [42]. Recall that the NAE3SAT problem asks whether a given Boolean formula in conjunctive normal form with clauses containing exactly three literals can be satisfied by a truth assignment of its variables, subject to the constraint that at least one literal in each clause is `false`. The proof in [8] also uses an easy-to-prove observation [32, 39] that states that if a graph admits some orientation without directed cycles and without transitive edges, it also admits one such orientation where an arbitrarily chosen node is the only sink. Hence, finding one such orientations where the resulting digraph is restricted to be a multi-source single-sink digraph (or equivalently a single-source multi-sink digraph) is NP-complete as well, even if the only sink (or the only source) is provided in advance. In this paper we address the problem of finding an orientation of an undirected graph such that the resulting digraph is a non-transitive st-graph, where both the vertices s and t are provided in advance. Namely we prove the NP-completeness of the following problem.

Problem: Non-Transitive st-Orientation (NTO)
Instance: An undirected graph $G = (V, E)$ and two vertices $s, t \in V$.
Question: Does there exist a non-transitive st-orientation of G?

It is not hard to see that the NTO problem is in NP, as one could non-deterministically choose among the two possible orientations of each edge in E and then check in polynomial time if the obtained orientation is a non-transitive st-orientation of G. To prove the hardness we have two possible strategies. The first strategy uses the result in [40] where it is shown that the NTO problem where nodes s and t are not provided in advance (recognizing "cover graphs of lattices" in the terminology of [40]) is NP-complete. We call this problem Relaxed-NTO.

Problem: Relaxed-Non-Transitive st-Orientation (Relaxed-NTO)
Instance: An undirected graph $G = (V, E)$.
Question: Does there exist a non-transitive st-orientation of G for some choice of two vertices $s, t \in V$?

We were unable to find a Karp-reduction from Relaxed-NTO to NTO. Specifically, we did not find a polynomial-time function that maps instances of Relaxed-NTO to instances of NTO while preserving the answer to the original instance of Relaxed-NTO. Instead, in Section 2.1 we describe a Turing-reduction, i.e., a process that uses a hypothetic deterministic Turing machine M that solves NTO in polynomial time to solve also in polynomial-time Relaxed-NTO, proving that the hypothetic machine M cannot exist unless P = NP.

The second strategy is motivated by the fact that the proof in [40] is rather complex. Indeed, similarly to [34], the proof in [40] starts from an instance of the coloring problem proved to be NP-complete by Lund and Yannakakis [29], and transforms such an instance into the instances of a sequence of different problems, the fourth of which is Relaxed-NTO. In [6], unaware of the result in [40], we presented a very simple Karp-reduction from NAE3SAT to NTO. This reduction is described in Section 2.2. Finally, Section 2.3 shows that NTO can be easily reduced to Relaxed-NTO. Hence, the two reductions of Sections 2.2 and 2.3 provide an alternative and simpler proof of the result in [40], as much as [8] provides an alternative and simpler proof of the result in [34]. Notably, both the reduction of Section 2.2 and the reduction in [8] start from an instance of NAE3SAT, although the constructions are quite different.

2.1 NP-Hardness of NTO by a Turing-reduction

Consider an instance $G = (V, E)$ of Relaxed-NTO and assume to have a deterministic Turing machine M that solves NTO. Choose in all the $O(|V|^2)$ possible ways the pair s, t and launch M on the obtained instances $\langle G, s, t\rangle$. It is immediate to see that if at least one instace $\langle G, s, t\rangle$ is a `Yes` instance then the instance $G = (V, E)$ is also a `Yes` instance. Conversely, if all the $\langle G, s, t\rangle$ instances are `No` instances then istance $G = (V, E)$ is a `No` instance. The above described process corresponds to a deterministic Turing machine M' that solves Relaxed-NTO. If the deterministic Turing machine M was able to decide NTO in polynomial time, as M' launches M a polynomial number (actually quadratic number) of times, M' would decide Relaxed-NTO also in polynomial time. This is a contradiction as in [40] Relaxed-NTO is shown to be NP-hard. Hence NTO cannot be solved in polynomial time.

2.2 NP-Hardness of NTO by a Karp-reduction

We reduce the following NP-complete problem [42].

Problem: Not-All-Equal 3SAT (NAE3SAT)
Instance: A Boolean formula that is a conjunction of clauses, where each clause is a disjunction of three literals from a set X of Boolean variables.
Question: Does there exist 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 φ, we construct an instance $I_\varphi = \langle G, s, t\rangle$ of NTO such that I_φ is a yes instance of NAE3SAT if and only if φ is a yes instance of NTO. Instance I_φ has one variable gadget V_x for each Boolean variable x and one clause gadget C_c for each clause c of φ. 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 [31]. Hence, G has to be assumed non-planar as well.

Before describing the gadgets, we introduce two simple observations on the constraints imposed by any non-transitive st-orientation of a graph G.

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 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: Consider a path $(v_1, v_2, \ldots, v_k)$ (refer to Fig. 2(a)). Suppose that in an st-orientation of G the edges (v_i, v_{i+1}), with $i = 1, \ldots, k-1$, are not all directed from v_i to v_{i+1} (as shown in Fig. 2(b)) and that they are not all directed from v_{i+1} to v_i. It follows that two edges of the path have an inconsistent orientation (as in Fig. 2(c)) and the path contains 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. □

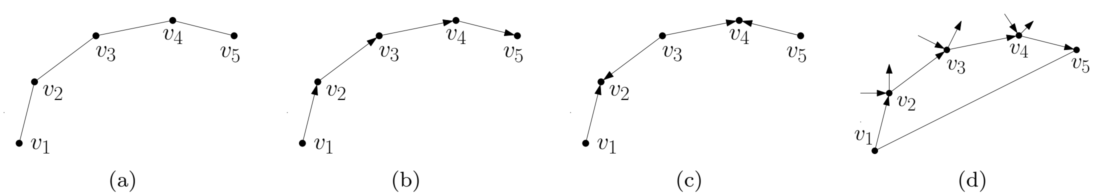


Figure 2: (a) A path of G with all internal vertices of degree two. (b) A consistent orientation of the path. (c) An inconsistent orientation of the path generates sinks or sources. (d) A directed path of G and a chord.

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. 2(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. □

The main ingredient of the reduction is the *fork gadget* (refer to Fig. 3), that is composed of ten edges $e_1, e_2, \ldots, e_{10}$, such that e_1, e_2, e_3, and e_4 have a common endpoint, denoted by v; e_5, e_6, and e_9 have a common endpoint, denoted by w; e_7, e_8, and e_{10} have a common andpoint, denoted

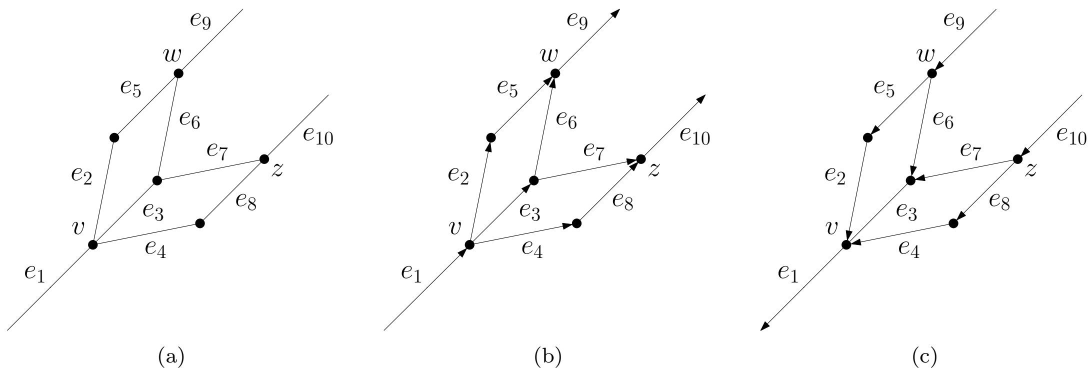


Figure 3: (a) The fork gadget. (b)-(c) The two possible orientations of the fork gadget in a non-transitive st-orientation of the whole graph.

by z; e_3, e_6, and e_7 have a common endpont; e_2 and e_5 have a common endpoint; and e_4 and e_8 have a common endpoint. The following lemma holds.

Lemma 1 *Let G be an undirected graph having a fork gadget F as an induced subgraph such that F 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. 3(b)). Either e_9 or e_{10} are oriented exiting F, since 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_5 must be both directed in the same direction. If they were directed towards v, then we would have a directed cycle (e_3, 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 contradicton 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 exiting 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 entering 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.

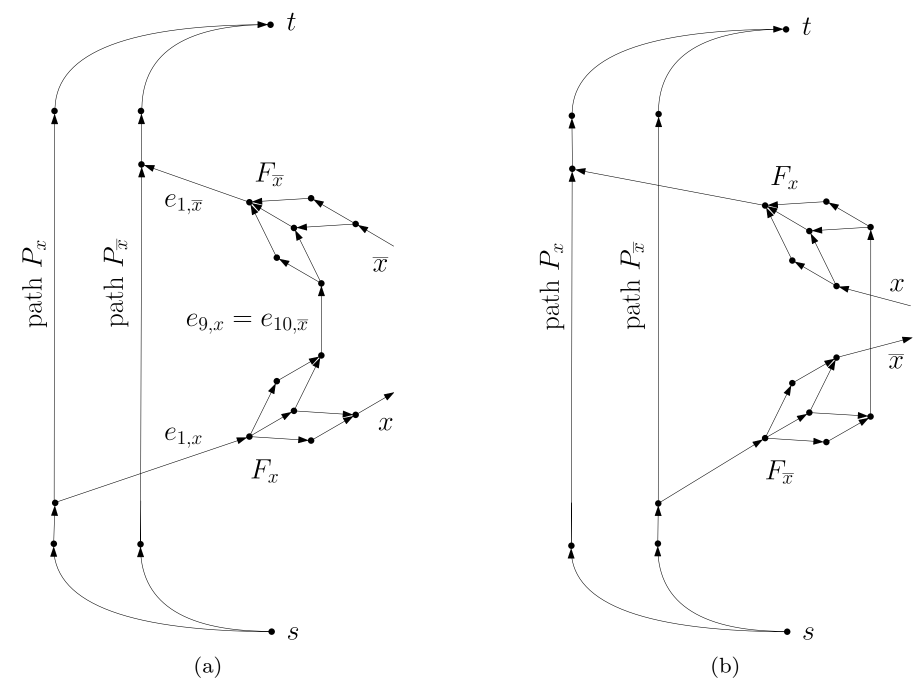


Figure 4: The variable gadget V_x and its `true` (a) and `false` (b) orientations.

With analogous and symmetric arguments it can be proved that if e_1 is oriented exiting F (refer to Fig. 3(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. 3(b) and 3(c) and the statement follows. □

For each Boolean variable x of ϕ we construct a *variable gadget* V_x by suitably combining two fork gadgets, denoted F_x and $F_{\overline{x}}$, as follows (see Fig. 4). 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 $F_{\overline{x}}$ is identified with edge e_9 of F_x. The two edges e_9 of $F_{\overline{x}}$ and e_{10} of F_x are denoted $\overline{x}$ and x, respectively. The construction of V_x is such that, even if a directed path was added outside V_x from edge x to edge $\overline{x}$ or vice versa, no directed cycle traverses V_x. In fact, in both the orientations depicted in Figs. 4(a) and 4(b) there is no directed path inside V_x from an entering edge to an exiting edge. Further, observe that, since the length of the two paths P_x and $P_{\overline{x}}$ is four, the edges of P_x and $P_{\overline{x}}$ cannot be transitive edges with respect to any directed path originating from s, ending with t, and traversing V_x. We have the following lemma.

Lemma 2 *Let G be an undirected graph containing a variable gadget V_x as an induced subgraph. 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.*

Proof: Suppose edge e_1 of F_x is oriented entering F_x (see edge $e_{1,x}$ of Fig. 4(a)). By Lemma 1 edge x is oriented exiting F_x and, hence, exiting V_x. Also edge e_9 of F_x, which coincides with

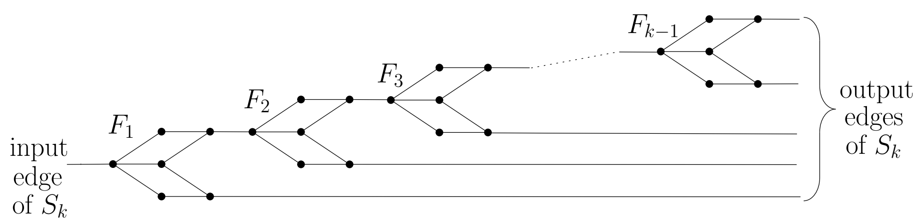


Figure 5: The split gadget S_k.

e_{10} of $F_{\overline{x}}$ (see the edge labeled $e_{9,x} = e_{10,\overline{x}}$ of Fig. 4(a)), is oriented exiting F_x and entering $F_{\overline{x}}$. Always by Lemma 1, edge e_1 of $F_{\overline{x}}$ is oriented exiting $F_{\overline{x}}$ (see edge $e_{1,\overline{x}}$ of Fig. 4(a)) and edge e_9 of $F_{\overline{x}}$, which coincides with edge $\overline{x}$ of V_x, is oriented entering $F_{\overline{x}}$ and, hence, entering V_x.

Suppose now that edge e_1 of F_x is oriented exiting F_x (see Fig. 4(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 $F_{\overline{x}}$, is oriented entering F_x and exiting $F_{\overline{x}}$. Now, always by Lemma 1, edge e_1 of $F_{\overline{x}}$ is oriented entering $F_{\overline{x}}$ and edge e_9 of $F_{\overline{x}}$, which coincides with edge $\overline{x}$ of V_x, is oriented exiting $F_{\overline{x}}$ and, hence, exiting V_x. □

By virtue of Lemma 2 we associate the `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. 4(a)). We call such an orientation the ***true** orientation of V_x*. Analogously, we associate the `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. 4(b)). Observe that edge x (edge $\overline{x}$, respectively) is oriented exiting V_x when the literal x (the literal $\overline{x}$, respectively) is `true`. Otherwise edge x (edge $\overline{x}$, respectively) is oriented entering V_x.

The *split gadget* S_k is composed of a chain of $k-1$ fork gadgets $F_1, F_2, \dots F_{k-1}$, where, for $i = 1, 2, \dots, k-2$, the edge e_9 of F_i is identified with the edge e_1 of F_{i+1}. We call *input edge of* S_k the edge denoted e_1 of F_1. Also, we call *output edges of* S_k the $k-1$ edges denoted e_{10} of the fork gadgets $F_1, F_2, \dots F_{k-1}$ and the edge e_9 of F_{k-1} (see Fig. 5). The next lemma is immediate and we omit the proof.

Lemma 3 *Let G be an undirected graph having a split gadget S_k as an induced subgraph such that S_k 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 ouput 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 $c = (l_1 \vee l_2 \vee l_3)$ 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. 6). We prove the following.

Theorem 1 *Problem* NTO *is NP-hard.*

Proof: The reduction from an instance φ of NAE3SAT to an instance I_φ previously described is performed in time linear in the size of φ.

Suppose $I_\varphi = \langle G, s, t\rangle$ is a positive instance of NTO and consider any non-transitive *st*-orientation of G_φ. Consider a clause c of φ and the corresponding vertex v_c in G. Since vertex v_c is

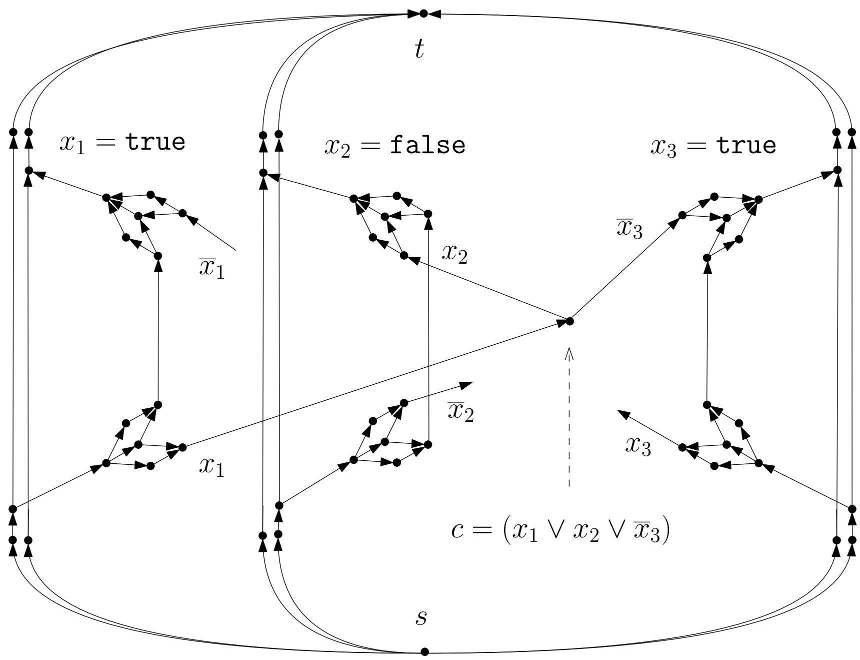


Figure 6: The clause gadget C_c for clause $c = (x_1 \vee x_2 \vee \overline{x}_3)$. The configurations of the three variable gadgets correspond to the truth values $x_1 = \texttt{true}$, $x_2 = \texttt{false}$, and $x_3 = \texttt{true}$. The clause is satisfied because the first literal x is `true` and the second and third literals x_2 and $\overline{x}_3$ are `false`.

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 φ is a yes instance.

Conversely, suppose that instance φ is a yes instance of NAE3SAT. Consider a truth assignment to the variables in X that satisfies φ. Orient the edges of each variable gadget V_x as depicted in Fig. 4(a) or Fig. 4(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, in the obtained orientation s is the only source and t is the only sink. Regarding acyclicity, observe that variable gadgets and clause gadgets whose edges are oriented as depicted in Fig. 4 and Fig. 6, 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, it can be easily checked that the obtained orientation of G is non-transitive. □

Observe that since instance I_φ used in the proof of Theorem 1 is biconnected, Problem NTO is NP-hard even on biconnected graphs.

2.3 Reduction of NTO to Relaxed-NTO

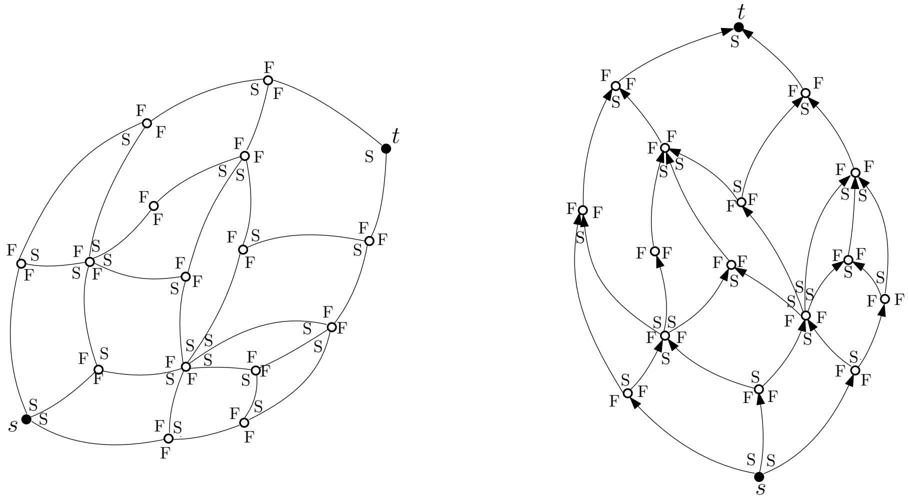
In this section we provide a reduction of NTO to Relaxed-NTO. Consider an instance $\langle G^*, s^*, t^* \rangle$ of NTO. Add two vertices s^+ and t^+ to G^* and connect them to s^* and to t^*, respectively. Call G^+ the obtained graph. Since s^+ and t^+ have degree one in G^+, in any non-transitive 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 of NTO to Relaxed-NTO, proving the hardness of the latter problem. Since Relaxed-NTO is also trivially in NP it is NP-complete.

3 ILP 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 problem.

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., [11]) 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 [24]). Note that, if e is a transitive edge in a given planar embedding of G', it remains transitive in any other planar embedding of G' (the property of being transitive is not related to planarity). Hence, the aforementioned property 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



(a) (b)

Figure 7: (a) An st-labeling of a plane graph G with prescribed nodes s and t. (b) The corresponding st-orientation of G.

v. As it was proved in [17], 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)[1], 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;

(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. 7 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

[1] Note that, a label F (flat) does not necessarily imply that the geometric angle will be a π angle; however we use this notation to be consistent with the one introduced in [17] and in other subsequent papers on the subject.

embedding of G with s and t on the external face) and that minimizes the number of transitive edges. The ILP 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 integer linear program uses a number of variables and constraints that is linear in the size of G; it is defined as follows.

Sets. Denote by V, E, and F the sets of vertices, edges, and faces of G, respectively. Also let $F_{\text{int}} \subset F$ 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 use 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 $y_{uvf} = 1$ then 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.

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

$$\sum_{v\in V(f)} x_{vf} = 2 \qquad \text{for } f \in F_{\text{int}} \tag{2}$$

$$\sum_{f\in F(v)} x_{vf} = \deg(v) - 2 \qquad \text{for } v \in V \setminus \{s,t\} \tag{3}$$

$$x_{sf} = 1 \qquad \text{for } f \in F_{\text{int}} \cap F(s) \tag{4}$$

$$x_{tf} = 1 \qquad \text{for } f \in F_{\text{int}} \cap F(t) \tag{5}$$

$$x_{uf} + x_{vf} \leq y_{uvf} + 1 \qquad \text{for } f \in F_{\text{int}} \text{ and } (u,v) \in E(f) \tag{6}$$

$$z_{uv} \geq y_{uvf} \qquad \text{for } e = (u,v) \in E \text{ and } f \in F(e) \tag{7}$$

$$x_{vf} \in \{0,1\} \quad y_{uvf} \in \{0,1\} \quad z_{uv} \in \mathbb{R} \tag{8}$$

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 real number; 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.

4 Experimental Analysis

We evaluated the efficiency of our ILP model using 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.

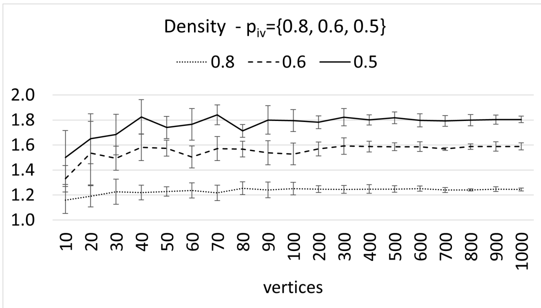
Instances. The experiments have been executed on a large benchmark of instances, each instance consisting of a plane biconnected graph and two vertices s and t on the external face. These graphs are randomly generated with the same approach used in previous experiments in graph drawing (see, e.g., [3]). 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 $p_{\rm iv}$ (which is a parameter of the generation process), while the Insert-Edge operation is performed with probability $1 - p_{\rm iv}$. For each operation, the elements (faces, vertices, or edges) involved are randomly selected with equal probability. 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 $(n, p_{\rm iv})$, with $n \in \{10, 20, \dots, 90, 100, 200, \dots, 900, 1000\}$ and $p_{\rm iv} \in \{0.2, 0.4, 0.5, 0.6, 0.8\}$, for a total of 950 graphs. Higher values of $p_{\rm iv}$ lead to sparser graphs.

On average, for $p_{\rm iv} = 0.8$ we have graphs with density of 1.23 (close to the density of a tree), for $p_{\rm iv} = 0.5$ we have graphs with density of 1.76, and for $p_{\rm iv} = 0.2$ we have graphs with density 2.53 (close to the density of maximal planar graphs). Fig. 8 shows for each sample the average density (number of edges divided by the number of vertices) of the graphs in that sample, together with the standard deviation. In addition to these information, Table 1 and Table 2 in the appendix report for each sample the minimum and maximum density values.

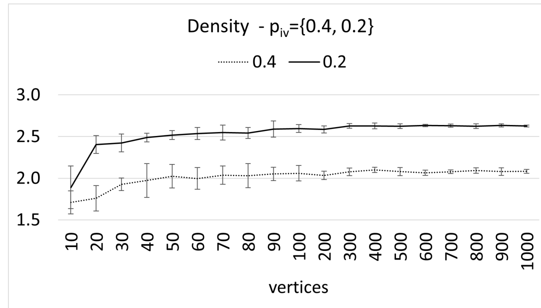
Experimental Goals. Our experimental analysis has three main goals:

- (G1) Evaluate the efficiency of our approach, i.e., the running time required by our ILP model. We call OptST the algorithm that solves the integer linear program;
- (G2) Evaluate the percentage of transitive edges in the solutions of the ILP model and how many transitive edges are saved with respect to applying a classical linear-time algorithm that computes an unconstrained st-orientation of the graph [22];
- (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 an algorithm that computes an st-orientation as a preliminary step.

For (G2) and (G3) we used implementations available in the GDToolkit library [10] 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 [22]. We refer to this

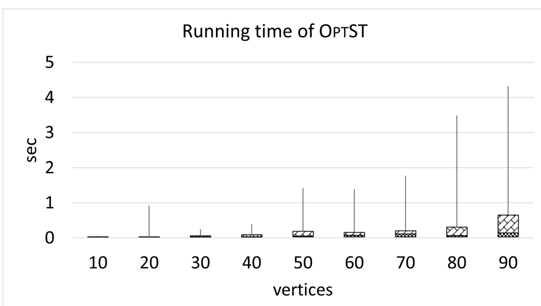


(a)

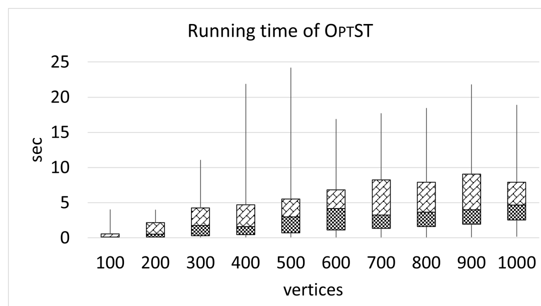


(b)

Figure 8: Density (mean values) and standard deviation of the different instances of our graph benchmark for: (a) $p_{\text{iv}} = 0.8 - 0.5$ and (b) $p_{\text{iv}} = 0.4 - 0.2$.



(a)



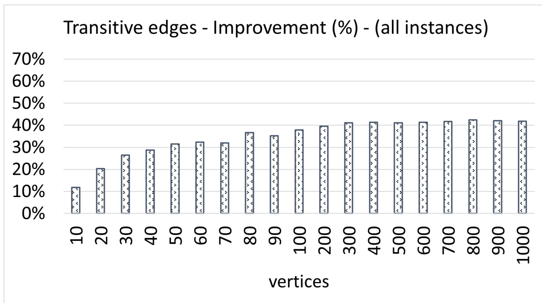
(b)

Figure 9: Box-plots of the running time of OPTST. Whiskers represent the minimum and the maximum values; for each box, the horizontal segment represents the median value, while the lower and the upper part represent the first and the third quartile, respectively.

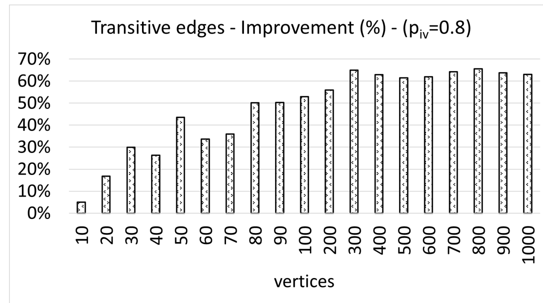
algorithm as BASEST. (*b*) A linear-time algorithm that first computes a visibility representation of an undirected planar graph based on a given *st*-orientation of the graph, and then computes from this representation a planar polyline drawing [12]. We call DRAWBASEST and DRAWOPTST the applications of this drawing algorithm to the *st*-graphs resulting from BASEST and OPTST, respectively.

Experimental Results. As for Goal (G1), Fig. 9 reports the running time (in seconds) of 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 up to 90 vertices and the other for the larger instances. OPTST is rather fast: 75% of the instances with up to 90 vertices are 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 a thousand vertices.

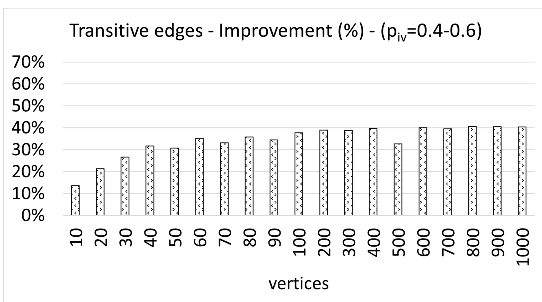
As for Goal (G2), Fig. 10 shows the reduction (in percentage) of the number of transitive edges



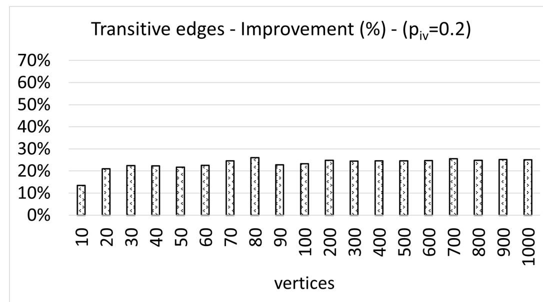
(a)



(b)



(c)

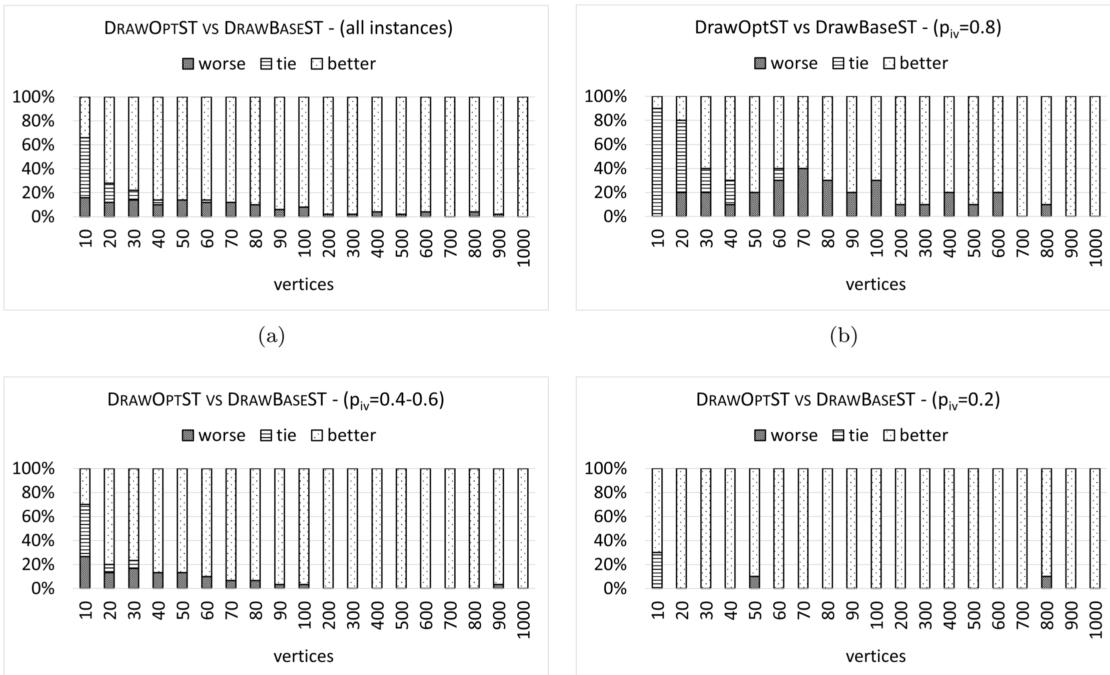


(d)

Figure 10: Improvement (%) in the number of transitive edges.

in the solutions of OptST with respect to the solutions of BaseST. More precisely, Fig. 10(a) reports values averaged over all instances with the same number of vertices; Fig. 10(b), Fig. 10(c), and Fig. 10(d) report the same data, partitioning the instances by different values of p_{iv}, namely 0.8 (the sparsest instances), 0.4-0.6 (instances of medium density), and 0.2 (the densest instances). For each instance, let trOpt and trHeur be the number of transitive edges of the solutions computed by OptST and BaseST, respectively. The reduction percentage of OptST against BaseST is measured by the value $\left(\frac{\text{trHeur}-\text{trOpt}}{\max\{1,\text{trHeur}\}}\times 100\right)$. 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 greater 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. For completeness, we also report in the appendix the total amount of transitive edges created by the two algorithms OptST and of BaseST, expressed both as absolute values and as percentages with respect to the total number of edges of the graph (Fig. 16). As expected, the amount of transitive edges increases with the density of the graph (in particular, denser graphs have a higher probability of containing triangles, hence transitive edges).

As for Goal (G3), Fig. 11 shows the percentage of instances for which DrawOptST produces drawings that are better than those produced by DrawBaseST in terms of area requirement (the label "better" of the legend). It can be seen that DrawOptST computes more compact drawings for the majority of the instances. In particular, it is interesting to observe that this is most often the case even for the densest instances (i.e., those for $p_{\text{iv}} = 0.2$), for which we have previously



(a) (b) (c) (d)

Figure 11: Instances for which DrawOptST produces drawings that are more compact than DrawBaseST (label "better").

seen that the average reduction of transitive edges is less evident. We also observe that in a small percentage of cases, when the graph is small or rather sparse, the reduction of transitive edges does not cause a reduction of the drawing area. We guess that this behavior may depend on the fact that for most of these instances the absolute number of transitive edges is small, also in the solution of the heuristic. The positive trend becomes definitely clear when the size and the density of the instances increase. For those instances for which DrawOptST computes more compact drawings than DrawBaseST, Fig. 12 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. 13 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 $p_{\mathrm{iv}} = 0.8$ and $p_{\mathrm{iv}} = 0.2$, the correlation between these two measures is quite evident. For the instances of medium density ($p_{\mathrm{iv}} \in \{0.4, 0.5, 0.6\}$), the highest values of improvement in terms of area requirement are observed for reductions of transitive edges between 22% and 66%. Figures 14 and 15 show examples of drawings computed by DrawBaseST and DrawOptST for two of our instances.

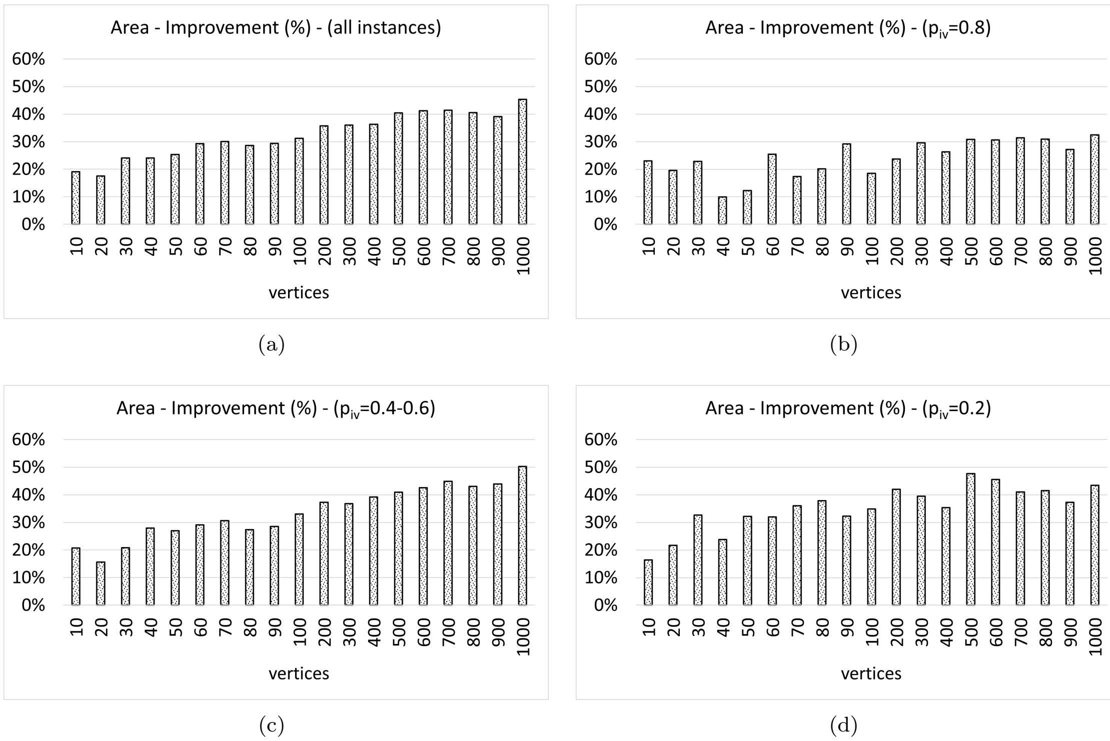


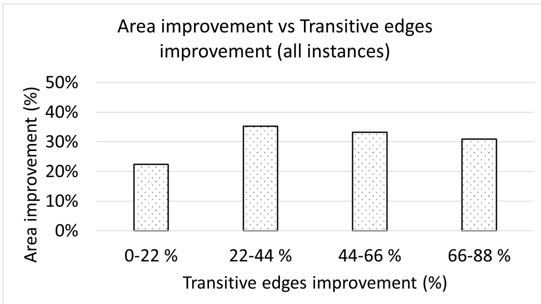
Figure 12: Area improvement (%) of DrawOptST with respect to DrawBaseST, for the instances where DrawOptST is "better" (i.e., the "better" instances in Fig. 11).

5 Final Remarks and Open Problems

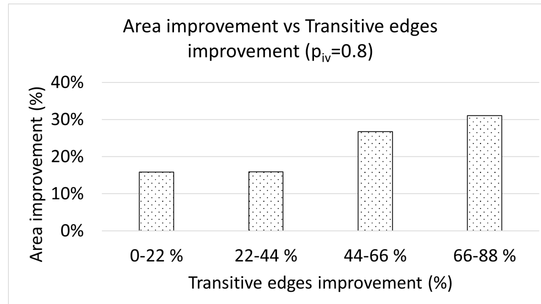
We addressed the problem of computing *st*-orientations with the minimum number of transitive edges. This problem has practical applications in graph drawing, as finding an *st*-orientation is at the heart of several graph drawing algorithms. Although *st*-orientations without transitive edges have been studied from a combinatorial perspective [24], there is a lack of practical algorithms, and 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 able to solve instances with up to 1000 vertices in about 10 seconds. It reduces on average by 35% the number of transitive edges with respect to 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 (with an improvement between 30% and 50%) in most cases (see also Fig. 14 and Fig. 15).

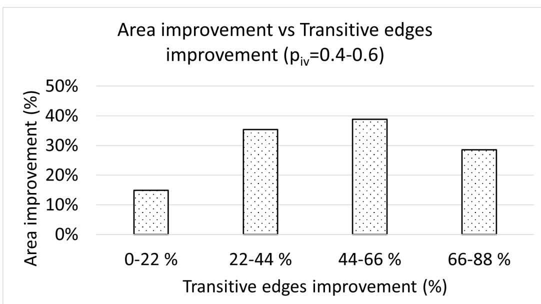
We conclude by suggesting two natural future research directions:



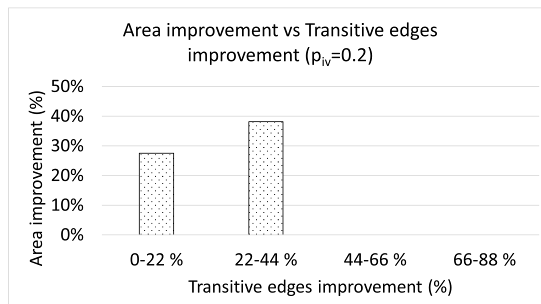
(a)



(b)



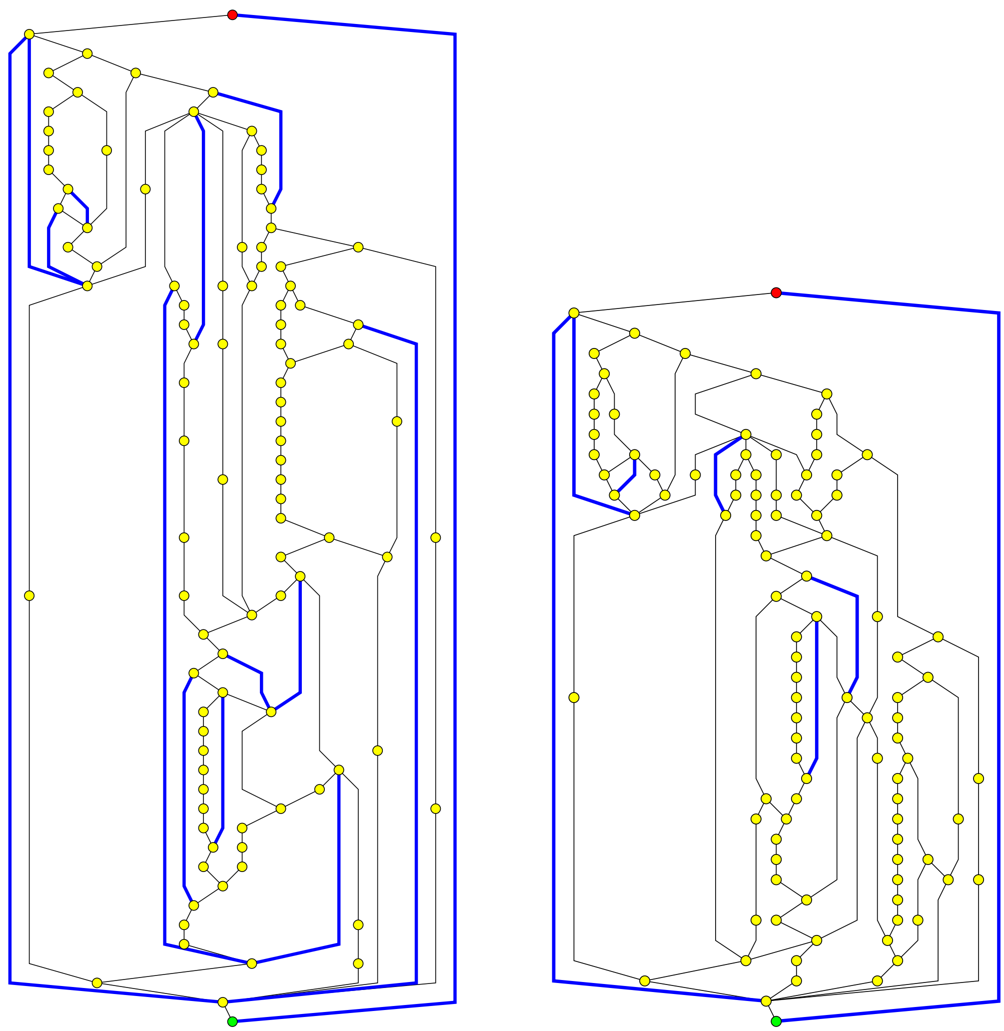
(c)



(d)

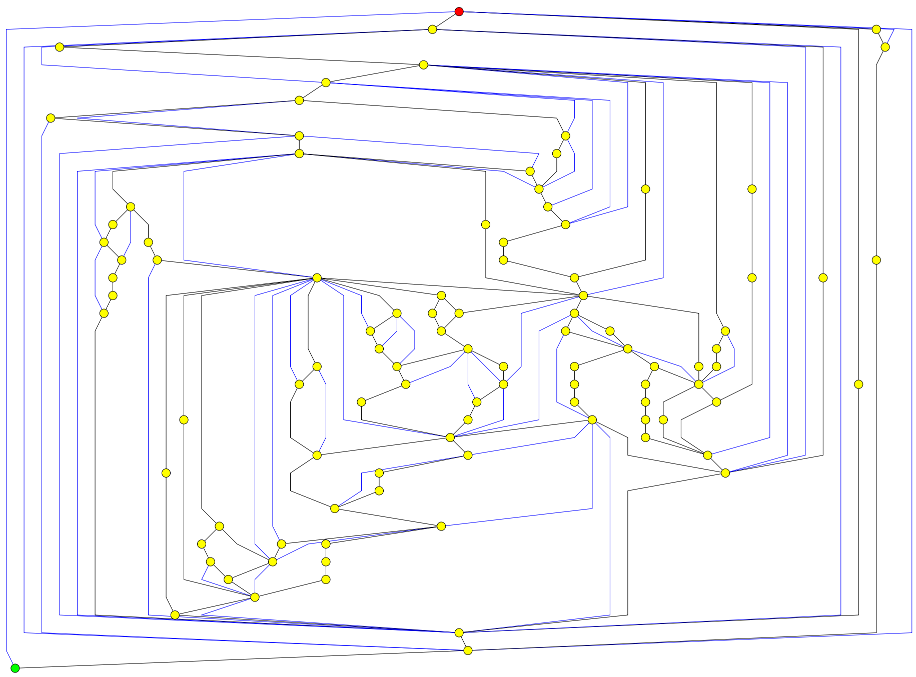
Figure 13: Correlation between the improvement (reduction) in terms of drawing area and in terms of transitive edges improvement.

- 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?
- From a practical point of view, it would be relevant to design fast heuristics for computing st-orientations of graphs with few transitive edges, and experiment their behavior on large real-world networks.

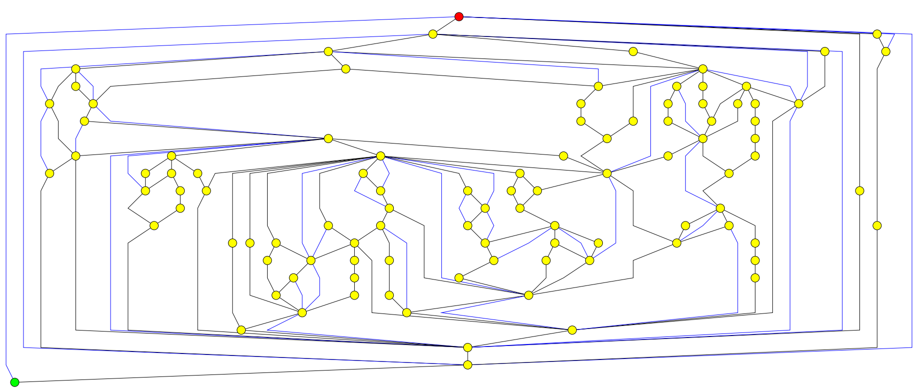


(a) 14 transitive edges (b) 7 transitive edges

Figure 14: Two polyline drawings of the same plane graph with 100 vertices and $p_{\mathrm{iv}} = 0.8$ computed by (a) DrawBaseST and (b) DrawOptST. Transitive edges are colored blue and are thicker.



(a) 52 transitive edges



(b) 37 transitive edges

Figure 15: Two polyline drawings of the same plane graph with 100 vertices and $p_{\text{iv}} = 0.5$ computed by (a) DRAWBASEST and (b) DRAWOPTST. Transitive edges are colored blue.

References

[1] M. Alzohairi and I. Rival. Series-parallel planar ordered sets have pagenumber two. In *Graph Drawing*, volume 1190 of *Lecture Notes in Computer Science*, pages 11–24. Springer, 1996. doi:10.1007/3-540-62495-3_34.

[2] P. Angelini, L. Cittadini, W. Didimo, F. Frati, G. Di Battista, M. Kaufmann, and A. Symvonis. On the perspectives opened by right angle crossing drawings. *J. Graph Algorithms Appl.*, 15(1):53–78, 2011. doi:10.7155/jgaa.00217.

[3] P. Bertolazzi, G. Di Battista, and W. Didimo. Computing orthogonal drawings with the minimum number of bends. *IEEE Trans. Computers*, 49(8):826–840, 2000. doi:10.1109/12.868028.

[4] C. Binucci and W. Didimo. Computing quasi-upward planar drawings of mixed graphs. *Comput. J.*, 59(1):133–150, 2016. doi:10.1093/comjnl/bxv082.

[5] C. Binucci, W. Didimo, and M. Patrignani. Upward and quasi-upward planarity testing of embedded mixed graphs. *Theor. Comput. Sci.*, 526:75–89, 2014. doi:10.1016/j.tcs.2014.01.015.

[6] C. Binucci, W. Didimo, and M. Patrignani. st-orientations with few transitive edges. In P. Angelini and R. von Hanxleden, editors, *Graph Drawing and Network Visualization (GD 2022)*, pages 201–216, Cham, 2023. Springer International Publishing. doi:10.1007/978-3-031-22203-0_15.

[7] U. Brandes. Eager st-ordering. In R. H. Möhring and R. Raman, editors, *Algorithms - ESA 2002, 10th Annual European Symposium*, volume 2461 of *Lecture Notes in Computer Science*, pages 247–256. Springer, 2002. doi:10.1007/3-540-45749-6_25.

[8] G. Brightwell. On the complexity of diagram testing. *Order*, 10(4):297–303, 1993. doi:0.1007/BF01108825.

[9] M. Chimani, C. Gutwenger, M. Jünger, G. W. Klau, K. Klein, and P. Mutzel. The open graph drawing framework (OGDF). In *Handbook of Graph Drawing and Visualization*, pages 543–569. Chapman and Hall/CRC, 2013.

[10] G. Di Battista and W. Didimo. Gdtoolkit. In *Handbook of Graph Drawing and Visualization*, pages 571–597. Chapman and Hall/CRC, 2013.

[11] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice-Hall, 1999.

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

[13] G. Di Battista, R. Tamassia, and I. G. Tollis. Area requirement and symmetry display of planar upward drawings. *Discret. Comput. Geom.*, 7:381–401, 1992. doi:10.1007/BF02187850.

[14] E. Di Giacomo, W. Didimo, G. Liotta, and S. K. Wismath. Book embeddability of series-parallel digraphs. *Algorithmica*, 45(4):531–547, 2006. doi:10.1007/s00453-005-1185-7.

[15] M. Dickerson, D. Eppstein, M. T. Goodrich, and J. Y. Meng. Confluent drawings: Visualizing non-planar diagrams in a planar way. *J. Graph Algorithms Appl.*, 9(1):31–52, 2005. doi:10.7155/jgaa.00099.

[16] W. Didimo. Upward graph drawing. In *Encyclopedia of Algorithms*, pages 2308–2312. 2016. doi:10.1007/978-1-4939-2864-4_653.

[17] W. Didimo and M. Pizzonia. Upward embeddings and orientations of undirected planar graphs. *J. Graph Algorithms Appl.*, 7(2):221–241, 2003. doi:10.7155/jgaa.00068.

[18] P. Eades, A. Symvonis, and S. Whitesides. Three-dimensional orthogonal graph drawing algorithms. *Discret. Appl. Math.*, 103(1-3):55–87, 2000. doi:10.1016/S0166-218X(00)00172-4.

[19] M. Eiglsperger, M. Kaufmann, and F. Eppinger. An approach for mixed upward planarization. *J. Graph Algorithms Appl.*, 7(2):203–220, 2003. doi:10.7155/jgaa.00067.

[20] D. Eppstein and J. A. Simons. Confluent Hasse diagrams. *J. Graph Algorithms Appl.*, 17(7):689–710, 2013. doi:10.7155/jgaa.00312.

[21] S. Even and R. E. Tarjan. Computing an *st*-numbering. *Theor. Comput. Sci.*, 2(3):339–344, 1976.

[22] S. Even and R. E. Tarjan. Corrigendum: Computing an *st*-numbering. TCS 2(1976):339-344. *Theor. Comput. Sci.*, 4(1):123, 1977.

[23] F. Frati, M. Kaufmann, J. Pach, C. D. Tóth, and D. R. Wood. On the upward planarity of mixed plane graphs. *J. Graph Algorithms Appl.*, 18(2):253–279, 2014. doi:10.7155/jgaa.00322.

[24] É. Fusy, E. Narmanli, and G. Schaeffer. On the enumeration of plane bipolar posets and transversal structures. *CoRR*, abs/2105.06955, 2021.

[25] P. Healy and N. S. Nikolov. Hierarchical drawing algorithms. In *Handbook of Graph Drawing and Visualization*, pages 409–453. Chapman and Hall/CRC, 2013.

[26] M. Jünger and P. Mutzel, editors. *Graph Drawing Software*. Springer, 2004. doi:10.1007/978-3-642-18638-7.

[27] M. Kaufmann and D. Wagner, editors. *Drawing Graphs, Methods and Models (the book grow out of a Dagstuhl Seminar, April 1999)*, volume 2025 of *Lecture Notes in Computer Science*. Springer, 2001. doi:10.1007/3-540-44969-8.

[28] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In *Theory of Graphs: Internat. Symposium (Rome 1966)*, pages 215–232, New York, 1967. Gordon and Breach.

[29] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. In *25th ACM Symposium on the Theory of Computing (STOC)*, pages 286–293, 1993. doi:10.1145/167088.167172.

[30] R. M. McConnell and J. P. Spinrad. Modular decomposition and transitive orientation. *Discret. Math.*, 201(1-3):189–241, 1999. doi:10.1016/S0012-365X(98)00319-7.

[31] B. M. E. Moret. Planar NAE3SAT is in P. *SIGACT News*, 19(2):51–54, 1988. doi:10.1145/49097.49099.

[32] K. M. Mosesian. Certain theorems on strongly basable graphs. *Akad. NaukArmian. SSR Dokl.*, 55:83–86, 1972. In Russian.

[33] J. Nešetřil and V. Rödl. Complexity of diagrams. *Order*, 3(4):321–330, 1987. doi:10.1007/BF00340774.

[34] J. Nešetřil and V. Rödl. More on the complexity of cover graphs. *Commentationes Mathematicae Universitatis Carolinae*, 36(2):269–278, 1995.

[35] O. Ore. *Theory of Graphs*, volume 38 of *American Mathematical Society Colloquium Publications*. American Mathematical Society, Providence, RI, 1962. doi:10.1090/coll/038.

[36] C. Papamanthou and I. G. Tollis. Algorithms for computing a parameterized *st*-orientation. *Theor. Comput. Sci.*, 408(2-3):224–240, 2008. doi:10.1016/j.tcs.2008.08.012.

[37] C. Papamanthou and I. G. Tollis. Applications of parameterized *st*-orientations. *J. Graph Algorithms Appl.*, 14(2):337–365, 2010. doi:10.7155/jgaa.00210.

[38] C. Platt. Planar lattices and planar graphs. *Journal of Combinatorial Theory, Series B*, 21(1):30–39, 1976. doi:10.1016/0095-8956(76)90024-1.

[39] O. Pretzel. On reorienting graphs by pushing down maximal vertices. *Order*, 3(2):135–153, 1986. doi:10.1007/BF00390104.

[40] V. Rödl and L. Thoma. The complexity of cover graph recognition for some varieties of finite lattices. *Order*, 12(4):351–374, 1995. doi:10.1007/BF01110379.

[41] P. Rosenstiehl and R. E. Tarjan. Rectilinear planar layouts and bipolar orientations of planar graphs. *Discret. Comput. Geom.*, 1:343–353, 1986. doi:10.1007/BF02187706.

[42] T. J. Schaefer. The complexity of satisfiability problems. In *Proc. of the 10th Annual ACM Symposium on Theory of Computing*, pages 216–226, 1978. doi:10.1145/800133.804350.

[43] J. Thostrup. Partial ordered sets and graphs. MSc. dissertation, Odense University, Denmark, 56 pp., 1992. In Danish.

[44] R. Wiese, M. Eiglsperger, and M. Kaufmann. *yFiles* - visualization and automatic layout of graphs. In *Graph Drawing Software*, pages 173–191. Springer, 2004. doi:10.1007/978-3-642-18638-7_8.

A Appendix

	0.8				0.6				0.5			
n	AVG	MIN	MAX	SD	AVG	MIN	MAX	SD	AVG	MIN	MAX	SD
10	1.16	1.00	1.40	0.11	1.33	1.10	1.50	0.11	1.50	1.20	1.80	0.22
20	1.19	1.05	1.30	0.08	1.54	1.30	2.15	0.25	1.65	1.35	2.05	0.20
30	1.23	1.07	1.37	0.10	1.49	1.37	1.67	0.10	1.68	1.43	1.93	0.16
40	1.22	1.10	1.30	0.06	1.58	1.43	1.78	0.11	1.83	1.58	2.08	0.14
50	1.22	1.16	1.28	0.04	1.57	1.46	1.66	0.06	1.74	1.54	1.86	0.09
60	1.24	1.15	1.33	0.06	1.51	1.38	1.63	0.09	1.77	1.55	1.95	0.13
70	1.22	1.16	1.36	0.06	1.57	1.41	1.71	0.10	1.84	1.66	1.93	0.08
80	1.25	1.19	1.33	0.05	1.57	1.49	1.68	0.06	1.71	1.63	1.79	0.05
90	1.24	1.16	1.33	0.06	1.54	1.40	1.71	0.10	1.80	1.67	1.96	0.11
100	1.25	1.15	1.34	0.05	1.53	1.40	1.67	0.09	1.80	1.69	1.97	0.09
200	1.25	1.20	1.28	0.03	1.57	1.50	1.65	0.06	1.78	1.69	1.84	0.05
300	1.25	1.19	1.30	0.03	1.59	1.48	1.67	0.07	1.82	1.73	1.93	0.07
400	1.25	1.19	1.31	0.03	1.59	1.53	1.64	0.04	1.80	1.74	1.86	0.04
500	1.25	1.21	1.27	0.03	1.59	1.53	1.62	0.03	1.82	1.75	1.89	0.05
600	1.25	1.21	1.29	0.02	1.59	1.54	1.64	0.04	1.80	1.73	1.88	0.05
700	1.24	1.21	1.27	0.02	1.57	1.55	1.59	0.01	1.79	1.71	1.84	0.04
800	1.24	1.23	1.26	0.01	1.59	1.55	1.62	0.02	1.80	1.73	1.88	0.05
900	1.25	1.22	1.28	0.02	1.59	1.54	1.66	0.04	1.80	1.75	1.86	0.04
1000	1.24	1.23	1.26	0.01	1.59	1.56	1.63	0.03	1.80	1.77	1.85	0.03

Table 1: Density of the different instances of our graph benchmark for $p_{\text{iv}} = 0.8 - 0.5$.

	0.4				0.2			
n	AVG	MIN	MAX	SD	AVG	MIN	MAX	SD
10	1.71	1.50	2.00	0.14	1.89	1.40	2.20	0.26
20	1.76	1.60	2.05	0.15	2.41	2.25	2.55	0.11
30	1.93	1.83	2.07	0.08	2.42	2.23	2.57	0.11
40	1.97	1.70	2.23	0.20	2.49	2.43	2.58	0.05
50	2.02	1.80	2.30	0.14	2.54	2.40	2.68	0.09
60	2.00	1.83	2.25	0.13	2.54	2.43	2.67	0.07
70	2.04	1.89	2.20	0.11	2.55	2.41	2.70	0.09
80	2.03	1.79	2.18	0.14	2.54	2.44	2.65	0.07
90	2.05	1.93	2.17	0.08	2.59	2.42	2.76	0.10
100	2.06	1.90	2.20	0.09	2.60	2.54	2.70	0.05
200	2.03	1.92	2.10	0.05	2.58	2.53	2.65	0.04
300	2.08	2.02	2.15	0.05	2.63	2.58	2.68	0.03
400	2.10	2.04	2.15	0.03	2.63	2.55	2.66	0.03
500	2.08	2.02	2.16	0.05	2.62	2.59	2.68	0.03
600	2.07	2.02	2.11	0.02	2.63	2.61	2.65	0.01
700	2.08	2.04	2.11	0.02	2.63	2.60	2.66	0.02
800	2.09	2.05	2.14	0.03	2.62	2.59	2.67	0.03
900	2.08	2.02	2.17	0.04	2.63	2.60	2.66	0.02
1000	2.08	2.05	2.12	0.02	2.63	2.61	2.64	0.01

Table 2: Density of the different instances of our graph benchmark for $p_{\text{iv}} = 0.4 - 0.2$

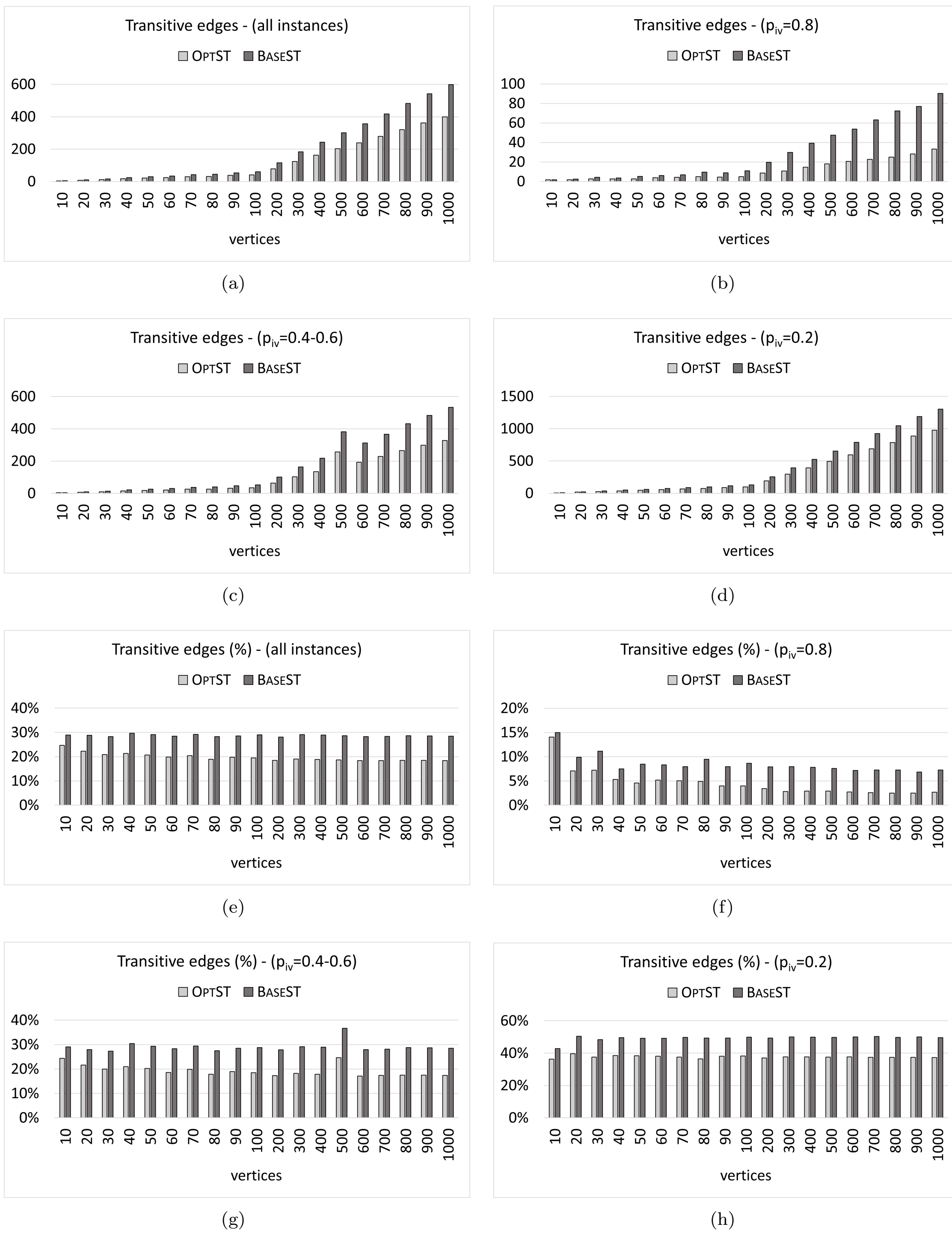


Figure 16: Amount of transitive edges in the solutions of OptST and BaseST: (a)–(d) Absolute values; (e)–(h) Percentages with respect to the total number of edges.