Theoretical Computer Science 1045 (2025) 115291

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Partial and constrained level planarity ☆

Guido Brückner [a], Ignaz Rutter [b,*]

[a] Faculty of Informatics, Karlsruhe Institute of Technology (KIT), Germany
[b] Faculty of Computer Science and Mathematics, University of Passau, Germany

ARTICLE INFO

Section Editor: Pinyan Lu

Handling Editor: Ralf Klasing

ABSTRACT

Let $G = (V, E)$ be a directed graph and $\ell : V \to [k] := \{1, 2, \dots, k\}$ a level assignment such that $\ell(u) < \ell(v)$ for all directed edges $(u, v) \in E$. A level-planar drawing of G maps each vertex v to a unique point on the horizontal line ℓ with y-coordinate $\ell(v)$ and each directed edge to a y-monotone Jordan arc between its endpoints such that no two arcs cross in their interior.

In the problem Constrained Level Planarity (CLP for short), we are further given a partial ordering $\lhd_i$ of $V_i := \ell^{-1}(i)$ for each $i \in [k]$, and we seek a level-planar drawing where the linear order $\prec_i$ of the vertices on ℓ_i is a linear extension of $\lhd_i$. A special case of this is the problem Partial Level Planarity (PLP for short), where we are asked to extend a given level-planar drawing $\mathcal{H}$ of a subgraph H of G to a complete drawing $\mathcal{G}$ of G without modifying the given drawing, i.e., the restriction of $\mathcal{G}$ to H must coincide with $\mathcal{H}$.

We give a simple polynomial-time algorithm with running time $O(n^5)$ for CLP of single-source graphs that is based on a simplified version of an existing level-planarity testing algorithm for single-source graphs. We introduce a modified type of PQ-tree data structure that is capable of efficiently handling the arising constraints to improve the running time to $O(n + ks)$, where s denotes the size of the constraints. We complement this result by showing that PLP is NP-complete even in very restricted cases. In particular, PLP remains NP-complete even when G has a constant number of levels, and when G is a subdivision of a triconnected planar graph with bounded degree.

1. Introduction

A *k-level graph* is a directed graph $G = (V, E)$ together with a leveling $\ell : V \to [k] := \{1, 2, \dots, k\}$ that assigns each vertex to one of k levels so that $\ell(u) < \ell(v)$ for each directed edge $(u, v) \in E$. Since most graphs occurring in this paper are level graphs, we will refer to them simply as graphs and explicitly mention if graphs are not level graphs. A *level drawing* of G (or simply a *drawing* of G when it is clear from the context that G is a level graph) maps each vertex v to a point on the horizontal line with y-coordinate $\ell(v)$ and each directed edge to a y-monotone Jordan arc between its endpoints. A crossing-free level drawing is *level planar*, and a level graph is *level planar* if it admits a level-planar drawing. The *level planarity testing problem* is to decide whether a given input graph is level planar or not.

In this paper, we study a generalization of the level planarity testing problem where, in addition to the k-level graph G, we are given *constraints* on the order of the vertices in each level in the form of a partial order $\lhd_i$ for each level $i \in [k]$. The task is

☆ A preliminary version of this article appeared as "Partial and Constrained Level Planarity" in *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA'17)*, pages 2000–2011.

* Corresponding author.
E-mail addresses: brueckner@kit.edu (G. Brückner), rutter@fim.uni-passau.de (I. Rutter).

https://doi.org/10.1016/j.tcs.2025.115291
Received 27 February 2024; Received in revised form 21 February 2025; Accepted 28 April 2025

Available online 6 May 2025

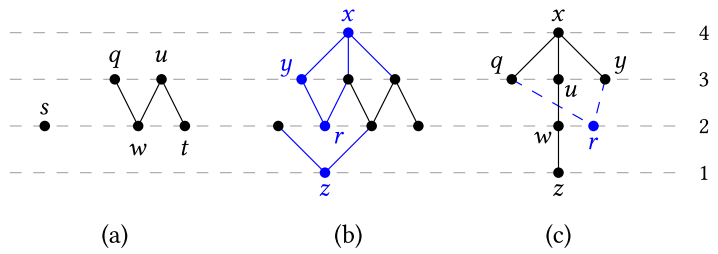

Fig. 1. A partial level-planar drawing (a) and an extension of it (b). The partial level-planar drawing shown in (c) does not have an extension. Namely, the vertex r and its incident edges cannot be inserted into the partial drawing without creating edge crossings.

to determine a level-planar drawing of G *compatible* with the constraints, i.e., such that the vertex order in each level i is a linear extension of $\lhd_i$. We call this problem Constrained Level Planarity or CLP for short.

A strongly related problem is the problem of extending a given drawing $\mathcal{H}$ of a subgraph H of G, also called *partial drawing*, to a level-planar drawing $\mathcal{G}$ of G without changing the given drawing $\mathcal{H}$. That is, the restriction of $\mathcal{G}$ to H must coincide with $\mathcal{H}$. We call such a drawing $\mathcal{G}$ an *extension*. The problem of deciding whether a given partial drawing admits an extension is called Partial Level Planarity, or PLP for short. See Fig. 1 that shows a partial drawing (a), an extension of it (b), and also a partial drawing that does not have an extension (c). Of course, taking for each level i the order $\lhd_i$ as the vertex order on level i in $\mathcal{H}$ yields a set of constraints for G. An extension is necessarily compatible with all these constraints. Later, we will see that for so-called proper level graphs, where all edges connect vertices of adjacent levels, these instances are actually equivalent. Since both problems can be reduced to the corresponding problem on proper level graphs, it turns out that indeed PLP can be seen as a special case of CLP.

Related work. Historically, level drawings were among the first drawing styles that were studied systematically with the introduction of the famous Sugiyama framework [57]. Level drawings are frequently used for visualizing hierarchical data and there is extensive research on every step of the framework; see [34] for a survey. Due to the importance of crossings on the visual clarity of drawings [52], the concept of level planarity has also received considerable attention.

The complexity of testing level planarity has been the subject of intensive research. Di Battista and Nardelli [23] gave the first efficient recognition algorithm for the subclass of proper single-source level graphs. Subsequently, there were attempts to generalize this result to the general case, i.e., non-proper level graphs with multiple sources [35] that however contained subtle mistakes [42]. Eventually, this line of research culminated in the work of Jünger and Leipert [41], who gave a linear-time algorithm for the general case. Initially, their algorithm was purely a test and did not output a corresponding level-planar embedding. They later improved their algorithm to also output a corresponding level-planar embedding, if it exists, in the same running time [40].

While the linear-time algorithm from the work of Jünger and Leipert answers the question about the complexity of level planarity testing, their algorithm is quite complicated and there are no practical implementations available. To address this, several papers proposed slower but simpler algorithms. Randerath et al. [53] gave a much simpler recognition algorithm for proper level graphs with running time $O(n^2)$, and Harrigan and Healy [33] formulated a quadratic-time recognition and embedding algorithm for proper level graphs. However, a recent work shows that both algorithms are flawed [26]. There is also a quadratic-time algorithm for general (non-proper) level planarity that is based on a Hanani-Tutte characterization [28].

Beyond that, also radial variants, where the levels are represented by concentric circles rather than horizontal lines, have been considered [7]. There also exists a Hanani-Tutte characterization for radial level planarity [27]. Recently, a connection between the algorithm due to Randerath et al. [53] and the Hanani-Tutte characterization due to Fulek et al. [27] was used to obtain a simple, efficient recognition algorithm for radial level-planar graphs [16]. Further variants on cylinders [14] and on the torus [2] have also been considered.

Constrained versions of graph representations and planarity variants are another type of widely studied problem as, for visualization purposes, it is often desirable to find visualizations that satisfy additional properties. Some constrained versions of level planarity have been considered. Harrigan and Healy [33] and Angelini et al. [3] both studied level planarity where vertices of the same level must be ordered consistently with different kinds of *constraint trees*. Klemz and Rote showed that deciding level planarity is NP-complete even in the severely constrained case where the orders of all vertices on all levels are fixed [47]. In real-world settings, sources of constraints include restrictions imposed by a user, e.g., by specifying the left-to-right order of some vertices or even fixing a part of the drawing, or as a consistency requirement stemming from the dynamic nature of a network, where one considers the restrictions imposed by stable parts of a network as constraints on the representation of the next snapshot. The latter leads to a *partial drawing extension* problem. More formally, a *partially drawn graph* is a triplet $(G, H, \mathcal{H})$, where G is a graph, H of G is a subgraph and $\mathcal{H}$ is a drawing of H. The partial drawing extension problem with input $(G, H, \mathcal{H})$ asks to complete the drawing $\mathcal{H}$ to a drawing $\mathcal{G}$ of G without modifying the given subdrawing of $\mathcal{H}$, i.e., the restriction of $\mathcal{G}$ to H is identical to $\mathcal{H}$. The partial drawing extension problem, and the related simultaneous drawing problem have received considerable attention in the last years.

Angelini et al. [5] gave a linear-time algorithm for partial drawing extension of *topological* planar drawings, where edges are represented by arbitrary, non-crossing Jordan arcs between their endpoints. In this case, the positive instances have also been characterized via forbidden subgraphs [39] and there exists a Hanani-Tutte characterization [55], which also leads to a polynomial-time algorithm. In contrast, for planar straight-line drawings, the partial drawing extension problem is NP-hard [51], though it becomes polynomial-time solvable if H is biconnected and $\mathcal{H}$ is a convex drawing [50]. Recently, partial upward planar drawings have been

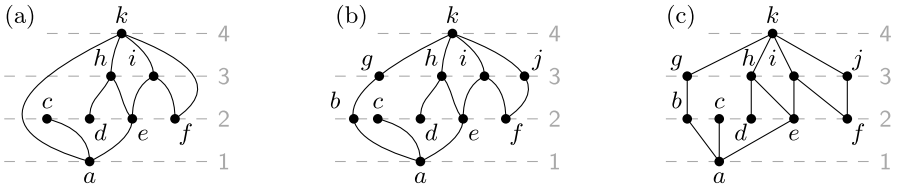

Fig. 2. A level-planar drawing of a non-proper level graph (a), its proper subdivision (b) and a combinatorially equivalent level-planar straight-line drawing (c).

considered [15,21]. Moreover, the problem has also been studied under the name *partial representation extension* for graph representations that are not node-link representations, in particular for different types of intersection representations, namely for interval representations [12,46], for proper and unit interval representations [44], for chordal graphs (which are intersection graphs of subtrees of a tree) [45], for permutation and function graphs [43] and, recently, for circle graphs [20] and trapezoid graphs [48], but also for contact representations [19] and hybrid representations [1].

We point out that the partial drawing extension problem is strongly related to the *simultaneous drawing problem*, whose input consists of two graphs G_1 and G_2 sharing a subgraph $G = G_1 \cap G_2$. The problem is to decide whether there exist drawings of G_1 and G_2 that coincide on G. This is equivalent to finding a drawing of G that extends to drawings of G_1 and G_2. It is known that the problem is NP-complete for planar straight-line drawings [24], though some special cases have recently been shown to be polynomial-time solvable [31]. For topological planar drawings, this problem is called SIMULTANEOUS EMBEDDING WITH FIXED EDGES and despite significant progress in the last years [4,6,8,9,11,12,29], the complexity status is still open; see [10,54] for surveys. The simultaneous representation problem for intersection representations was introduced by Jampani and Lubiw, who studied permutation and chordal graphs [38] and interval graphs [37]. Bläsius and Rutter [12] improved the running time for simultaneous interval representations to linear. Recently, level planarity has been considered in this setting, too; simultaneous level planarity is NP-complete for two graphs on three levels, as well as for three graphs on two levels, an efficient algorithm exists only for two graphs on two levels [2].

Contribution and outline. We study the level planarity problem subject to constraints and in the context of partial drawings. We present preliminaries in Section 2. In particular, we show that, indeed, PLP reduces to CLP. In Section 3 we present a simplified version of the algorithm of Di Battista and Nardelli [23] for the single-source case. We then introduce the order graph, a data structure that allows us to efficiently intersect an implicit representation of the set of all level-planar drawings of a graph with an implicit representation of a set of (not necessarily planar) level drawings that satisfy the constraints. Since the latter set is guaranteed to contain all level-planar drawings that satisfy the constraints, this allows us to solve CLP for single-source graphs in time $O(n^5)$. We then refine the algorithm and develop a modified version of a PQ-tree, which we call *constrained PQ-tree*. It handles both the planarity and the constraints in the same data structure. With this modification the algorithm runs in time $O(n + ks)$ for proper k-level graphs, where s is the *size* of the constraints, i.e., the number of distinct vertex pairs that are related by some $\lhd_i$. In Section 4 we complement these results by showing that PLP is NP-complete even in quite restricted cases, in particular, even when it has a constant number of levels, and when it is a subdivision of a triconnected planar graph (i.e., its combinatorial embedding is essentially fixed), the graph has fixed maximum degree and the graph has only a constant number of sources per level. This shows that, unless $P \neq NP$, it is unlikely that efficient algorithms can be achieved for much more general types of instances.

Moreover, while a Hanani-Tutte-style characterization exists for level planarity, our hardness result rules out the existence of such a characterization for partial level planarity. This contrasts the situation for usual (topological) planarity, where Hanani-Tutte characterizations exist for planarity and for partial planarity [55].

2. Preliminaries

A k-level graph $G = (V, E)$ is *proper* if $\ell(v) = \ell(u) + 1$ for each directed edge $(u, v) \in E$. Every k-level graph G can be transformed into a proper k-level graph $G' = (V', E')$ by subdividing edges. Likewise, a (not necessarily planar) level drawing Γ of G can be transformed into a level drawing Γ' of G'. See Fig. 2 (a) for a drawing of a non-proper level graph and (b) for its proper subdivision. In the following, we therefore restrict our treatment to proper level graphs. This is without loss of generality, though the resulting proper graph may have size in $\Theta(kn)$, where $n = |V|$ is the number of vertices of the original graph. Note that we may assume without loss of generality that each level contains at least one vertex, and therefore $k \leq n$.

Combinatorially, a level drawing Γ of a proper k-level graph $G = (V, E)$ can be described by a set $\prec = \{\prec_i\}_{i=1}^{k}$ of linear orderings, where $\prec_i$ is the linear ordering of the vertices in V_i along the horizontal line ℓ_i with y-coordinate i. For brevity, we sometimes omit the index and write $u \prec v$ if $u \prec_i v$ holds for two vertices u, v on the same level i.

For example, in Fig. 2 (b) and (c), $\prec_i$ coincides with the alphabetical order for each level i, i.e., these drawings are combinatorially equivalent. Di Battista and Nardelli give the following characterization of level-planar drawings.

Lemma 1 *([23, Lemma 1]). Let $G = (V, E)$ be a proper k-level graph and let $\prec = \{\prec_i\}_{i=1}^{k}$ be a drawing of G. Then $\prec$ is level planar if and only if for any distinct vertices $u, w \in V_j$ and $v, x \in V_{j+1}$, $j \in [k-1]$ with $(u, v), (w, x) \in E$ we have $u \prec_j w \Leftrightarrow v \prec_{j+1} x$.*

Two vertex pairs ss' with $s, s' \in V_i$ and tt' with $t, t' \in V_j$ are *equivalent* ($ss' \equiv tt'$ for short) if in every level-planar drawing $\prec$ of G it is either $s \prec s'$ and $t \prec t'$ or $s' \prec s$ and $t' \prec t$. Lemma 1 allows us to determine a first set of equivalent vertex pairs.

Of course, if $ss' \equiv tt'$ and $tt' \equiv uu'$, transitivity implies $ss' \equiv uu'$. More generally, we can obtain additional equivalent pairs by considering suitable paths in G as follows. Let $\mathrm{lace}(a, b)$ denote the set of all paths whose first vertex is in level a, whose last vertex is in level b, and whose interior vertices v satisfy $a \leq \ell(v) \leq b$.

Lemma 2 *([23, Lemma 2]). Let G be a proper k-level graph and let $(s, \dots, t)$ and $(s', \dots, t')$ be two vertex-disjoint paths in* $\mathrm{lace}(a, b)$ *with $1 \leq a < b \leq k$. Then $ss' \equiv tt'$.*

Drawing constraints for a proper k-level graph $G = (V, E)$ are expressed as a set of partial orders $\lhd = \{\lhd_i\}_{i=1}^{k}$, where $\lhd_i$ is a partial order of V_i. A drawing $\prec$ of G *satisfies* the constraints $\lhd$ if and only if for every $\prec_i$ is a linear extension of $\lhd_i$ for each $i \in [k]$. The *size* s of $\lhd$ is the number of distinct vertex pairs that are related by $\lhd$. In general, s is quadratic in n. Similar to above, we sometimes omit the index and write $u \lhd v$ if $u \lhd_i v$ holds for two vertices on the same level i.

Let $(G, H, \mathcal{H})$ be an instance of PLP. The proper subdivision of $\mathcal{H}$ induces a total left-to-right order of the vertices on each level in the proper subdivision of H, i.e., an instance of CLP. Note that, while storing a partial order on r elements may generally require $\Omega(r^2)$ storage, if we simply store the linear order on each level, the storage required for representing a partial drawing of a proper level is linear in n. Moreover, Lemma 1 gives that in a straight-line level-planar drawing of a proper graph, the vertices can be moved horizontally without creating crossings as long as their order within the levels does not change. Thus, if a drawing satisfying the order constraints of an instance of PLP exists, the vertices of the drawing can always be moved so that the partial drawing is preserved. It follows that PLP is indeed a special case of CLP.

Lemma 3. *For proper level graphs, PLP reduces to CLP in linear time.*

Note that this uses the fact that $\mathcal{H}$ is a drawing. A combinatorial embedding of H would not be sufficient, because testing level planarity of non-proper level graphs where the orders of all vertices on all levels are fixed is NP-complete [47]. The complete edge-vertex order can therefore be assumed to be encoded in $\mathcal{H}$, which implies that PLP reduces to CLP in linear time even for non-proper graphs.

Conversely, drawing constraints are strictly more powerful than partial drawings. For example, with drawing constraints it is possible to express the constraints $u \lhd v, u \lhd w$ without deciding whether $v \lhd w$ or $w \lhd v$. This is not possible with partial drawings, since a partial drawing defines for each level i a total order $\lhd_i$ on the level-i vertices of H.

PQ-trees. Booth and Lueker [13] introduced the *PQ-tree* data structure, which is capable of representing sets of linear orders on a ground set. A PQ-tree is a rooted ordered tree, i.e., each inner node is equipped with a linear ordering of its children. For each node Y of a PQ-tree denote by $\mathrm{pertinent}(Y)$ the set of leaves of the subtree rooted at Y. Each inner node is either a P-node or a Q-node, and the leaves are in one-to-one correspondence with the elements of a finite ground set U, called the *yield* of T, i.e., $U = \mathrm{yield}(T)$. Each inner node has at least two children, so the size of T is linear in $|U|$. An in-order traversal of the leaves of T (e.g., by starting a DFS at an arbitrary leaf and visiting the neighbors of each vertex according to its cyclic ordering) gives a linear order of the elements in U, called the *frontier of* T and denoted by $\mathrm{frontier}(T)$.

There are two equivalence transformations for PQ-trees. The order of the neighbors of a P-node may be arbitrarily rearranged, whereas the order of the neighbors of a Q-node may be reversed. Performing such an equivalence transformation on a PQ-tree T results in a new PQ-tree T', which generally has a different frontier. Two PQ-trees T and T' are *equivalent*, i.e., $T \equiv T'$, if and only if T' can be obtained from T by a number of equivalence transformations. The set of *consistent permutations* represented by T is defined as $\mathrm{consistent}(T) = \{\mathrm{frontier}(T') \mid T' \equiv T\}$.

The usefulness of PQ-trees arises from the operation $\mathrm{update}(T, S)$. For a PQ-tree T and a subset $S \subseteq \mathrm{yield}(T)$, this operation returns a new PQ-tree T' that limits the linear orders represented by T to those in which the elements in S appear consecutively. If there are no such cyclic orders, the result is the *null tree*, which formally represents the empty set of orderings of a ground set, i.e., if T' is the null tree, then $\mathrm{consistent}(T') = \emptyset$.

The original update operation of Booth and Lueker [13] is based on a complicated application of several types of transformation templates. We use the implementation of Shih and Hsu [56], who generalized PQ-trees to so-called PC-trees that represent circular orders of a ground set, and showed that they can be used as a drop-in replacement for PQ-trees by adding a special root element to the tree and splitting the cyclic orders at that element into linear orders. In the following we use and extend the simplified version of PC-trees by Hsu and McConnel [36].

The update operation consists of three steps; see Fig. 3 for an illustration. The first step is to label all leaves $v \in S$ as *black* and all other nodes as *white*, and then iteratively label inner nodes as black if all its neighbors except one are black. An edge e of T is called *terminal* if both connected trees obtained by removing e from T contain black and white leaves. If $\mathrm{consistent}(T)$ contains no order in which the elements of S appear consecutively, the result is the null tree. Otherwise, the terminal edges form a path in T, the *terminal path*. In the second step, a new Q-node x is created and the edges of the terminal path are deleted. Then, for each node y on the terminal path, a split node y' is created, all black neighbors are moved from y to y', and y and y' are connected to x. Finally, in the third step, all edges from x to adjacent Q-nodes are contracted. As a cleanup, leaves that do not correspond to an element of the ground set are removed, and internal degree-2 nodes are contracted to obtain a well-defined PC-tree. We refer to [25] for details of the implementation.

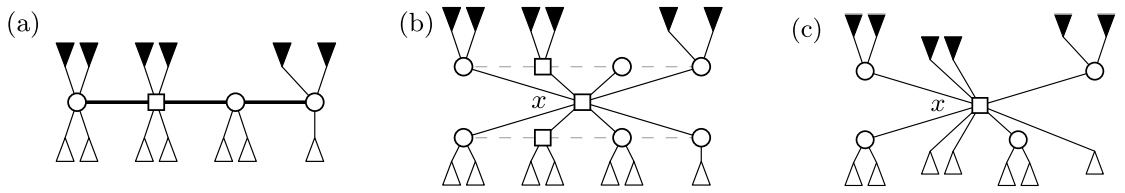

Fig. 3. The three-step update procedure for PQ-trees; Q-nodes are represented by squares, P-nodes by disks. (a) The terminal path is bold, the subtrees consisting of only black and of only white vertices are represented by black and white triangles, respectively. (b) The terminal path is split and a new Q-node x is connected to all vertices of the terminal path. (c) Contraction of edges between x and adjacent Q-nodes and cleanup.

Lemma 4 *([36, Lemma 4.7]). The* update(T, S) *operation can be performed in $O(p + |S|)$ time, where p is the length of the terminal path in T.*

3. Single-source graphs

Di Battista and Nardelli [23] presented a linear-time algorithm to test single-source proper level graphs for level planarity. In this section, we first present a simplified variant of their algorithm with the same running time. We then extend this algorithm to a CLP algorithm with running time $O(n^5)$ by introducing the new concept of *order graphs*. Finally, we introduce *constrained PQ-trees*, which are modified PQ-trees that are capable of efficiently maintaining additional constraints. With the use of constrained PQ-trees, we improve the running time of our algorithm to $O(n + ks)$, where k is the number of levels and s is the size of the constraints $\lhd$.

3.1. A simple level planarity testing algorithm for single-source graphs

For a proper k-level planar graph, we denote by G_j the subgraph induced by the vertices on levels $1, \dots, j$. Moreover, by $E_j = (V_j \times V_{j+1}) \cap E$, we denote the edges of G between levels j and $j+1$. We start out by making the following observation about level-planar drawings.

Lemma 5. *Let G be a proper k-level graph together with a level planar drawing $\prec$. Let $j \in [k-1]$ and for some $\lambda \in (0,1)$ let the order of edges in E_j in which they intersect with the horizontal line $y = j + \lambda$ be $(u_1, v_1) \prec (u_2, v_2) \prec \dots \prec (u_t, v_t)$. Then the edge endpoints are ordered consecutively, i.e., if $u_a = u_b$ with $1 \le a < b \le t$, it follows that $u_a = u_{a+1} = \dots = u_b$, and if $v_a = v_b$ with $1 \le a < b \le t$, it follows that $v_a = v_{a+1} = \dots = v_b$.*

Proof. Assume that $\prec$ is planar and that there exists $u_c \neq u_a = u_b$ with $a < c < b$. Then it is $(u_a, v_a) \prec (u_c, v_c) \prec (u_b = u_a, v_b)$. From $u_a \neq u_c \neq u_b$ it follows that either $u_a \prec u_c$ or $u_c \prec u_b$ holds true. If it is $u_a \prec u_c$, then (u_c, v_c) intersects with $(u_b = u_a, v_b)$. Conversely, if it is $u_c \prec u_a$, then (u_c, v_c) intersects with (u_a, v_a). With Lemma 1, this contradicts the planarity of $\prec$, so no such u_c can exist. The same idea can be used to show that there exists no $v_c \neq v_a$ with $a < c < b$. □

The idea of Di Battista and Nardelli is to perform a sweep of the single-source proper k-level graph, successively visiting each level $j \in [k]$ in increasing order. When going from level j to level $j+1$, they compute how level-planar drawings of G_j can be extended to level-planar drawings of G_{j+1}. To efficiently represent all possible planar drawings simultaneously, they use PQ-trees.

Our algorithm follows this approach, computing the same PQ-trees, albeit in a simpler way. Namely, Di Battista and Nardelli compute their PQ-tree by starting from a spanning tree of G, and thus their tree contains leaves that lie on multiple levels. By contrast, we start with a PQ-tree that contains only the source and then extend this tree level by level. In particular, for each of the trees we compute, all leaves lie on the same level.

Consider how a planar drawing $\prec$ of G_j can be extended to a drawing of G_{j+1}. Start by drawing non-intersecting edge stubs protruding from u for each edge $(u, v) \in E_j$. Lemma 5 states that in any planar drawing, these edge stubs have to be ordered so that identical endpoints are consecutive. The stubs can then be extended to complete edges meeting at their shared endpoints without causing any edge crossings.

The algorithm is as follows. Instead of considering individual drawings, for each level $j \in [k]$ a PQ-tree T_j is computed that represents the orders of level-j vertices across all level-planar drawings of G_j; see Fig. 4 for an example.

The tree T_1 consists of a single leaf, the source of G. Given a tree T_j, the tree T_{j+1} for the subsequent level is generated as follows. If T_j is the null tree, so is T_{j+1}. If T_j is not the null tree, consider each leaf u of T_j. If u has no outgoing edges in E_j, mark u as obsolete. Otherwise, add each outgoing edge as a child of u in T_j. Once all leaves have been considered, post-order traverse through T_j and mark any node as obsolete if all its children are obsolete. Then remove all obsolete nodes from T_j. The tree now represents the possible orders of the edges between levels j and $j+1$. For each vertex $v \in V_{j+1}$ we execute the following two steps. First, we make the edge leaves with endpoint v consecutive in T_j using the update operation. Recalling the PQ-tree update operation, the edge leafs are now in a single black subtree or in consecutive black subtrees attached to a Q-node. In the second step, we replace these black subtrees by a single leaf v. The following lemma establishes the correctness of this algorithm.

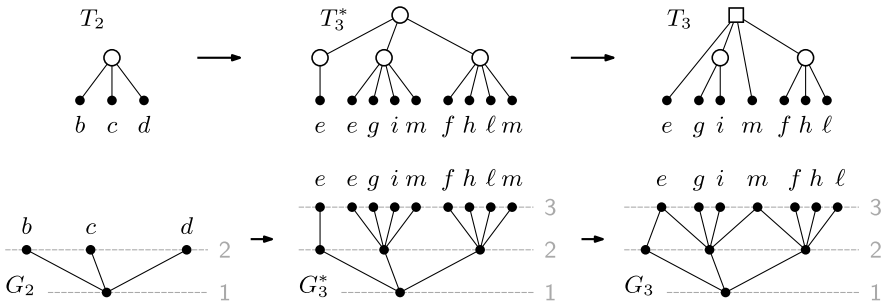

Fig. 4. Illustration of Di Battista and Nardelli's algorithm applied to the graph G_3. Starting from the PQ-tree T_2 in the left, the PQ-tree $T_3^\star$ in the middle is generated by adding all level-3 neighbors as children to b, c, d. Then, multiple occurrences are made consecutive and replaced by a single vertex, leading to T_3, as shown in the right.

Lemma 6. *Let G be a single-source proper k-level graph and let T_j be the PQ-tree for some level $j \in [k]$. Then $(v_1, v_2, \dots, v_t) \in \text{consistent}(T_j)$ if and only if there exists a level-planar drawing $\prec$ of G_j that satisfies $v_1 \prec_j v_2 \prec_j \dots \prec_j v_t$.*

Proof. We prove the claim by induction on the number of levels. Since G has a single source, we have $|V_1| = 1$ and the holds trivially for $j = 1$. Assume that the claim holds for some $j \in [k-1]$. We show that it also holds for $j+1$.

By the inductive hypothesis, $\text{consistent}(T_j)$ contains exactly the orders of level-j vertices across all level-planar drawings of G_j. We say that T_j *represents* G_j for short. Let $G_{j+1}^\star$ denote the graph obtained from G_{j+1} by splitting the vertices on level $j+1$ into multiple copies such that they all have degree 1 and label each copy with the corresponding original vertex on level $j+1$; see Fig. 4. By Lemma 5 the level-planar drawings of G_{j+1} correspond bijectively to the level-planar drawings of $G_{j+1}^\star$ where the copies of level $j+1$ with the same label are consecutive.

Let $T_{j+1}^\star$ be the PQ-tree obtained from T_j by turning each leaf that corresponds to a level-j vertex u into a P-node with a leaf for every edge in E_j incident to u, or removing u if no edge in E_j is incident to u. Note that the edges in E_j correspond bijectively with the level-$(j+1)$ vertices of $G_{j+1}^\star$. We first use Lemma 5 to show that $T_{j+1}^\star$ represents $G_{j+1}^\star$. Namely, any drawing $\prec$ of $G_{j+1}^\star$ induces also a drawing of G_j, whose ordering of level j vertices is represented by T_j. By Lemma 5 the level-$(j+1)$ vertices of $G_{j+1}^\star$ adjacent to the same vertex on level j are all consecutive in $\prec$. Therefore, a tree equivalent to $T_{j+1}^\star$ whose frontier coincides with the given drawing can be obtained by adding the leaves to T_j in that ordering. Conversely, by construction, any ordering represented by $T_{j+1}^\star$ induces a unique ordering for T_j, and a corresponding drawing of $G_{j+1}^\star$ can be obtained by adding the level-$(j+1)$ vertices to a corresponding drawing of G_j.

It follows that after applying the update operations, the resulting PQ-tree represents the level-$(j+1)$ vertex orders of all level-planar drawings of $G_{j+1}^\star$ where level-$(j+1)$-copies with the same label are consecutive. Since these correspond bijectively to the orders of the level-planar drawings of G_{j+1}, the PQ-tree T_{j+1} where all leaves with the same label v are replaced by a single leaf v for each $v \in V_{j+1}$ indeed represents the orders of vertices on level $j+1$ across all planar drawings of G_{j+1}. □

From Lemma 6 it follows that if T_j is the null tree for any $j \in [k]$, then G is not level planar. Otherwise, if T_k is not the null tree, G is level planar. For a family of sets $\mathcal{S} = \{S_1, S_2, \dots, S_t\}$ all update(T, S_i) operations are feasible in a total running time of $O(|T| + \sum_{i=1}^t |S_i|)$ [13]. Let n_j denote the number of level-j vertices. Since every inner node of T_j has at least two children, we have $|T_j| \in O(n_j + n_{j+1})$ and since $\mathcal{S}$ is a family of disjoint sets $\sum_{i=1}^t |S_i| \in O(n_{j+1})$. This gives linear running time for the entire algorithm. Di Battista and Nardelli also give an important link of inner nodes of the PQ-tree T_j to parts of the graph G_j; see Fig. 5 for an illustration.

Lemma 7 *([23], [49, Lemma 4.5]). Let G be a single-source proper level graph and let T_j be the PQ-tree for some level $j \in [k]$. Every P-node Y in T_j corresponds to a cutvertex of G_j whose removal from G_j separates vertices that appear in subtrees rooted at distinct children of Y. Every Q-node Y in T_j corresponds to a maximal biconnected component H in G_j. Further, there exists a cycle C in H so that each child c of Y corresponds to a cutvertex x of G_j that lies on C and such that removing x from G_j separates the vertices that appear in the yield of the subtree rooted at c from H.*

3.2. A polynomial-time CLP algorithm for single-source graphs

The algorithm from the previous section uses PQ-trees T_j to restrict all drawings of a level graph to just those that are level planar, represented by $\text{consistent}(T_j)$. This section introduces order graphs P_j, which are used to restrict all drawings of a level graph to just those that are compatible with the constraints $\lhd$ represented by $\text{consistent}(P_j)$, which are a partial order of the vertices.

More precisely, let $G = (V, E)$ be a proper k-level graph. The directed graph $\tilde{P}_j = (V, F)$ for G_j, $j \in [k]$, is constructed by taking the empty digraph on the same vertex set as G_j. Edges are then added step by step. First, for any vertex pair $u, v \in V_i$ on some level $i \in [j]$

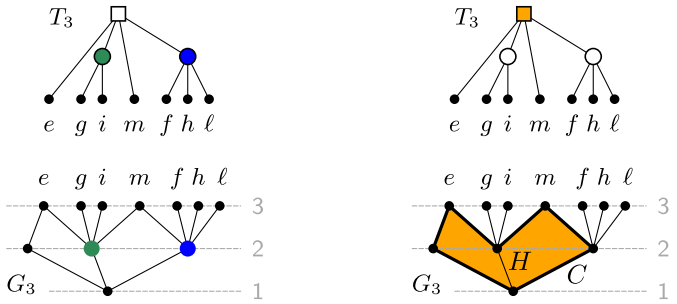

Fig. 5. Illustration of Lemma 7. On the left, the cutvertices of G_3 and the corresponding P-nodes in the PQ-tree are marked. In the right, a biconnected component H and its corresponding Q-node in the PQ-tree are shaded. The cycle C bounding H is bold.

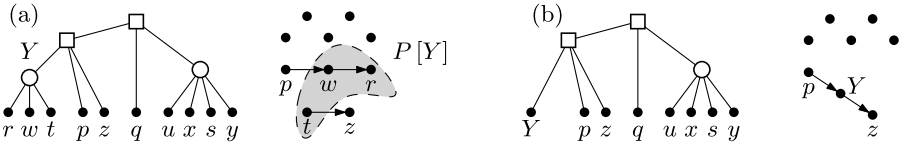

Fig. 6. Illustration of Lemma 9. Part (a) shows a PQ-tree T and an order graph P with $\text{yield}(P) = \text{yield}(T)$, and (b) shows $T \circ Y$ and $P \circ Y$ for an internal node Y of T whose children are all leaves.

with $u \lhd v$, the directed edge (u, v) is added to $\tilde{P}_j$. Next, following the necessary conditions from Lemma 1, whenever $(u, w) \in F$ and $uw \equiv vx$, then the directed edge (v, x) is added to $\tilde{P}_j$. To ensure transitivity, if $(u, v), (v, w) \in F$, then (u, w) is added to $\tilde{P}_j$ as well. These last two steps are repeated until no more edges can be added to $\tilde{P}_j$. We define the order graph $P_j := \tilde{P}_j[V_j]$ as the subgraph of $\tilde{P}_j$ induced by the vertices on level j. Similarly to the PQ-tree for level j, we denote by $\text{yield}(P_j) = V_j$ the vertex set of P_j, and by $\text{consistent}(P_j)$ the set of topological orderings of P_j.

We claim that the PQ-trees from the previous sections and the order graphs together encode all drawings of a level graph that are planar *and* satisfy the given constraints $\lhd$. More precisely, let $\text{consistent}(T_j, P_j) := \text{consistent}(T_j) \cap \text{consistent}(P_j)$. Our goal is to show that there exists a level-planar drawing of G compatible with $\lhd$ if and only if $\text{consistent}(T_j, P_j) \neq \emptyset$ for all $j \in [k]$ and to use this to test the existence of such a drawing.

The remainder of this section describes (i) the necessary algorithmic ingredients, i.e., construction of the order graph and the algorithm for testing whether $\text{consistent}(T, P) \neq \emptyset$ for a PQ-tree T and an order graph P, (ii) the interoperation of PQ-trees and order graphs, and (iii) the resulting CLP algorithm. We start with the algorithmic ingredients.

Lemma 8. *Let $(G, \lhd)$ be a constrained single-source proper k-level graph with n vertices. The order graphs $P_1, \dots, P_k$ can be computed in $O(n^5)$ time.*

Proof. The graphs $\tilde{P}_i$ can be constructed iteratively, starting with $\tilde{P}_1$, which consists of a single vertex and can be constructed in $O(1)$ time. Given $\tilde{P}_i$, the graph $\tilde{P}_{i+1}$ can be constructed by first adding the edges corresponding to $\lhd$ for vertices in V_{i+1}; this takes time linear in the size of $\lhd$ over all iterations. Adding edges whose existence is implied by Lemma 1 is possible by considering all $O(n^2)$ pairs of edges in E. The other edges can be added by running a transitive closure algorithm, e.g., the Floyd-Warshall algorithm [58], which requires $O(n^3)$ time. These last two steps are repeated exhaustively; the resulting graph is $\tilde{P}_{i+1}$.

For the running time, observe that the last two steps are executed at most $O(n^2)$ times over all iterations, because $\tilde{P}_k$ can contain at most n^2 edges. This gives an overall running time of $O(n^5)$ for constructing all graphs $\tilde{P}_1, \dots, \tilde{P}_k$. Clearly, $P_1, \dots, P_k$ can be computed from these in the same running time. □

The next step is to determine whether a PQ-tree T and an order graph P on the same vertex set are consistent, i.e., whether $\text{consistent}(T, P) \neq \emptyset$. Here we rely on an algorithm of Klavík et al. [46]. For the sake of completeness, we give a brief sketch of their consistency procedure. First, it picks an inner node Y of T that has only leaves as children. If $P[Y] := P\left[\text{yield}(\text{pertinent}(Y))\right]$ has no suitable topological ordering, T and P are inconsistent. Otherwise, the PQ-tree $T \circ Y$ is generated from T by removing all children of Y, making it a leaf, and the order graph $P \circ Y$ is generated from P by identifying the vertices in $\text{yield}(\text{pertinent}(Y))$ into a single vertex Y; see Fig. 6. Then T and P are consistent if and only if $T \circ Y$ and $P \circ Y$ are consistent. The running time is linear in the size of the order graph.

Lemma 9 *([46, Proposition 2.4]). Let T be a PQ-tree and P an order graph with identical yields. It can be tested whether* $\text{consistent}(T, P) \neq \emptyset$ *in time linear in the size of P.*

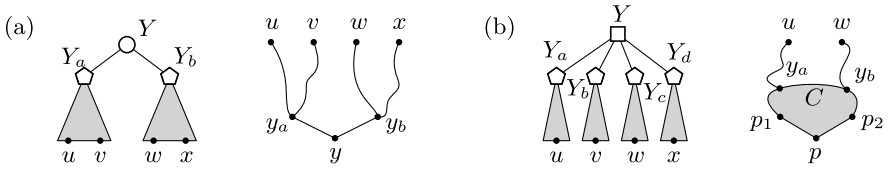

Fig. 7. Proof of Lemma 10 for P-nodes and Q-nodes in (a) and (b), respectively.

Note that while the above sketch destroys T by calculating $T \circ Y$, one can simply treat Y as a leaf in the recursion. The consistency procedure then applies equivalence transformations on T to make its frontier a topological ordering of P, i.e., reordering T according to P.

Consider now a single-source proper k-level graph $(G, \lhd)$. Let $T_1, \dots, T_k$ be the corresponding PQ-trees and let $P_1, \dots, P_k$ be the corresponding order graphs. Our goal is to show that G admits a level-planar drawing satisfying $\lhd$ if and only if $\text{consistent}(T_j, P_j) \neq \emptyset$ for each $j \in [k]$. The following lemma establishes a link between inner nodes of the PQ-tree and the relation $\equiv$ encoded in the order graphs.

Lemma 10. *Let G be a planar single-source proper level graph and let T be the corresponding PQ-tree for level j. Further, let Y be a node in T with ordered children $Y_1, Y_2, \dots, Y_t$ and let*

$$u \in \text{yield}(\text{pertinent}(Y_a)), \qquad w \in \text{yield}(\text{pertinent}(Y_b)),$$
$$v \in \text{yield}(\text{pertinent}(Y_c)), \qquad x \in \text{yield}(\text{pertinent}(Y_d)),$$

where $1 \leq a < b \leq t$ and $1 \leq c < d \leq t$. If Y is a Q-node, or if Y is a P-node with $c = a$ and $d = b$, then $uw \equiv vx$.

Proof. First, assume that Y is a P-node, as shown in Fig. 7 (a). Lemma 7 gives that Y corresponds to a cutvertex y in G, which is hence contained both in p_1 and in p_2. Let y_a be the neighbor of y that corresponds to Y_a and let y_b be the neighbor of y that corresponds to Y_b. Since G has a single source, there exist directed paths $(y_a, \dots, u)$ and $(y_b, \dots, w)$ in G. Lemma 2 yields $uw \equiv y_a y_b$. The same idea yields $vx \equiv y_a y_b$, which gives $uw \equiv vx$.

Now assume that Y is a Q-node, as shown in Fig. 7 (b). Lemma 7 gives that Y corresponds to a biconnected component H in G_j. Let $y_a, y_b \in V(H)$ denote the cutvertices of G_j that correspond to Y_a and Y_b, respectively. As above, there exist directed paths p_a, p_b from y_a, y_b to u, w, respectively.

Consider now a planar drawing of G_j whose ordering on level j corresponds with the $\text{frontier}(T_j)$. In this drawing H is bounded by a simple cycle C that visits y_a before y_b in clockwise direction. Since G has a single source, H contains a unique *peak* p, i.e., a vertex of minimum level. Let p_1 be the vertex that lies clockwise after p on C and let p_2 be the vertex that lies clockwise before p on C. Note that $p_1 \neq p_2$. Let $q_a = (p_1, \dots, y_a, \dots, u)$ be the path that is the concatenation of the clockwise subpath of C from p_1 to y_a and the path p_a. Similarly, let $q_c = (p_2, \dots, y_b, \dots, w)$ be the path that is the concatenation of the counterclockwise subpath of C from p_2 to y_b and the path p_b. Since p does not lie on q_a or q_b and p is the unique peak of H, p_1 and p_2 are the peaks of paths q_a and q_b, respectively. Therefore Lemma 2 gives $uw \equiv p_1 p_2$. Applying the same idea to the cutvertices y_c, y_d and vertices v, x yields $vx \equiv p_1 p_2$, and therefore $uw \equiv vx$. □

We are now ready to prove our main result. First observe that a constrained single-source proper level graph $(G, \lhd)$ can have drawing $\prec$ that is compatible with $\lhd$ only if $\text{consistent}(P_j, T_j) \neq \emptyset$ for each $j \in [k]$. To show that the converse holds as well, we prove a slightly stronger statement, namely that $\text{consistent}(T_k, P_k)$ contains exactly those linear-orders of the level-k-vertices of G which can occur in a planar drawing of G that is compatible with $\lhd$.

Lemma 11. *Let $(G, \lhd)$ be a constrained single-source k-level graph, and let T_j and P_j for $j \in [k]$ be the PQ-tree and the order graph of G_j, respectively. Assume further that $\text{consistent}(T_j, P_j) \neq \emptyset$ for $j = 1, \dots, k$ and let $\pi = (v_1, \dots, v_t)$ be a linear ordering of V_k. Then G has a level-planar drawing $\prec$ that is compatible with $\lhd$ and for which $v_1 \prec_k v_2 \prec_k \dots \prec_k v_t$ if and only if $\pi \in \text{consistent}(T_k, P_k)$.*

Proof. We first prove the necessity. Assume that there exists a level-planar drawing $\prec$ of G that is compatible with $\lhd$ such that $\prec_k = \pi$. Because $\prec$ is level-planar, Lemma 6 gives $\pi \in \text{consistent}(T_k)$. Since $\prec$ is compatible with $\lhd$, and the order graph P_k is constructed by adding only necessary edges (i.e., edges implied by transitivity or by Lemma 1), it follows that $\pi \in \text{consistent}(P_k)$. Together, this gives $\pi \in \text{consistent}(T_k, P_k)$.

We prove the reverse direction by induction on the number k of levels. Observe that the claim trivially holds for $k = 1$, since then G consists of a single vertex. For the inductive step, assume that the statement holds for all graphs on up to k levels and consider a graph G on $k+1$ levels. Let $\pi \in \text{consistent}(T_{k+1}, P_{k+1})$ be a linear ordering of V_{k+1}. We say that an ordering π' of the vertices on level k is *compatible with* π if ordering the vertices on levels k and $k+1$ as π' and π, respectively, yields a crossing-free drawing of the edges between levels k and $k+1$. We have the following claim.

Claim 1. *There is a linear ordering $\pi' \in \text{consistent}(T_k, P_k)$ that is compatible with π.*

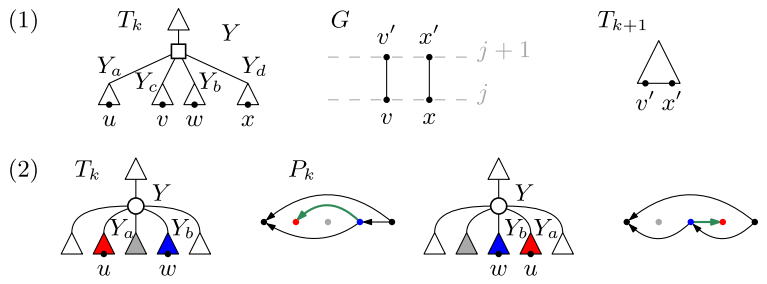

Fig. 8. Case (1) and (2) of the proof of Lemma 11. In case (1), suppose that vertices u, w appear in that order in the frontier of T_j and that there are vertices v, x with $uw \equiv vx$ that have distinct neighbors v', x' on level $j+1$. Then the frontier of T_{j+1} cannot lie in consistent(P_{j+1}). In case (2), find a conflict induced by a closest pair of reversed vertices, e.g., the green conflict between the children Y_a and Y_b (left). Then this conflict can be resolved without creating new conflicts (right). (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

We first assume the claim and show that it implies the lemma. Hence let $\pi' \in \text{consistent}(T_k, P_k)$ be compatible with π. Consider the constrained k-level subgraph $(G_k, \lhd_k)$ of G, where $\lhd_k$ is the restriction of $\lhd$ to vertices of the first k levels. Note that T_j, $j \in [k]$ are the PQ-trees of G_k and that P_j, $j \in [k]$ are the order graphs of G_k. By the inductive hypothesis, there is a drawing $\prec_k$ of G_k that is compatible with $\lhd_k$ and where the vertices on level k have the order π'. We then obtain the claimed drawing $\prec_{k+1}$ by adding the vertices of G on level $k+1$ in the order π. Since $\pi \in \text{consistent}(P_{k+1})$ and $\prec_k$ is compatible with $\lhd_k$, it follows that $\prec_{k+1}$ is compatible with $\lhd$.

It remains to prove the claim. To this end let $\pi = (v_1, v_2, \dots, v_t) \in \text{consistent}(T_{k+1}, P_{k+1})$. Lemma 6 gives the existence of a planar drawing $\prec_{k+1}$ of G with $v_1 \prec_{k+1} v_2 \prec_{k+1} \cdots \prec_{k+1} v_t$. Let $\pi'' = (u_1, u_2, \dots, u_{t'})$ be the order of the level-k vertices induced by $\prec_{k+1}$. Because $\prec_{k+1}$ is level-planar, it follows that π'' is compatible with π and by Lemma 6 that $\pi'' \in \text{consistent}(T_k)$. Hence there exists at least one permutation in consistent(T_k) that is compatible with π.

Let π'' be a linear ordering of the vertices on level k. A *conflict* is a pair u, w of vertices such that u occurs before w in π'', but P_k contains the edge (w, u). We choose $\pi' \in \text{consistent}(T_k)$ so that it is compatible with π and among all such choices it minimizes the number of conflicts. In the following we show that π' is conflict-free, and therefore π' satisfies the properties of the claim.

Assume for the sake of contradiction that π' has a conflict. Let u, w be a conflict and among all such pair assume that the number of vertices between u and w in π' is minimum. The order of u and w in $\text{frontier}(T_k)$ is decided by some internal node Y of T_k where u and w are leaves of subtrees rooted at distinct children of Y, say Y_a and Y_b; i.e., Y_a appears to the left of Y_b in the ordered sequence of children of Y. We distinguish cases based on whether Y is a Q- or a P-node.

1. First consider the case that Y is a Q-node. See Fig. 8 (1).
   (a) Suppose that there are level-k vertices v, x that (i) are leaves of subtrees rooted at distinct children of Y, say Y_c and Y_d, where Y_c appears to the left of Y_d, and (ii) have distinct neighbors v', x' on level $k+1$ in G_{k+1}. By definition $v \prec_{\pi'} x$, and since π' is compatible with π, it follows from Lemma 1 that $v' \prec_\pi x'$. On the other hand, Lemma 10 gives $uw \equiv vx \equiv v'x'$, and hence the arc (w, u) in P_k implies the arc (x', v') in P_{k+1}. But then $v' \prec_\pi x'$ contradicts $\pi \in \text{consistent}(P_{k+1})$.
   (b) Otherwise, reverse the order of the children of Y. Let T_k' denote the new tree. We show that $\text{frontier}(T_k')$ is compatible with π. Suppose drawing the edges between levels k and $k+1$ causes a crossing, i.e., there exists a crossing between two edges, say (v, v') and (x, x'), where $\ell(v) = \ell(x) = k$ and $\ell(v') = \ell(x') = k+1$. Since the crossing is produced by reversing the order of the children, it follows that reversing the order of the children of Y reverses the order of v and x, i.e., they are leaves of distinct children of Y. Moreover, $v' \neq x'$ since adjacent edges do not cross. This means that the assumptions from case 1a are fulfilled, a contradiction. Therefore, reversing the order of the children of Y does not cause a crossing, i.e., $\text{frontier}(T_k')$ is compatible with π and it has fewer conflicts than π'. This contradicts the choice of π'.
2. Second, consider the case that Y is a P-node.
   (a) Suppose that there are level-k vertices v, x that (i) are leaves of the subtrees rooted at Y_a and Y_b, respectively, and (ii) have distinct neighbors v', x' on level $k+1$ in G_{k+1}. Planarity gives $v' \prec_{k+1} x'$. On the other hand, Lemma 10 gives $uw \equiv vx \equiv v'x'$. Hence the arc (w, u) in P_k implies the arc (x', v') in P_{k+1}. But then $v' \prec_\pi x'$ contradicts $\pi \in \text{consistent}(P_{k+1})$.
   (b) Otherwise, we show how to reduce the conflicts in π'. Because we are not in case 2a, all leaves in the subtrees rooted at Y_a, Y_b have no neighbors on level $k+1$ or a single neighbor x' on level $k+1$ that is shared by all of them. We now distinguish two cases based on whether both Y_a and Y_b have a leaf that is adjacent to x'.
      i. Suppose that there exist leaves v, x in the subtrees rooted at Y_a, Y_b, respectively, that are both adjacent to x'. Since π' is compatible with π, it follows that all leaves of subtrees rooted at Y_i with $a \le i \le b$ have x' as their only neighbor on level $k+1$ or they do not have any neighbor on level $k+1$ in G_{k+1}. Let T_k' be the PQ-tree obtained from T_k by moving Y_a so that it becomes the right neighbor of Y_b. Note that this only changes the relative order of vertices that are leaves of subtrees rooted at children Y_i with $a < i \le b$. Because all of these vertices have the same neighbor x' on level $k+1$, if at all, the resulting ordering $\text{frontier}(T_k')$ is still compatible with π'. Finally, since u, w was a closest pair of conflicting vertices, $\text{frontier}(T_k')$ contains no new conflicts.
      ii. Otherwise, the leaves of one or both of the subtrees rooted at Y_a, Y_b have no neighbor on level $k+1$. Assume that the leaves of Y_a have no such neighbor. Similarly to above, we obtain the PQ-tree T_k' from T_k by moving Y_a so that

it becomes the right neighbor of Y_b. This only changes the relative order of vertices that are leaves of subtrees rooted at Y_a and children Y_i with $a < i \leq b$. Note that since Y_a has no leaf that has a neighbor on level $k+1$, $\mathrm{frontier}(T'_k)$ is compatible with π. Moreover, u, w is not a conflicting pair of $\mathrm{frontier}(T'_k)$ and since u, w was a closest pair of conflicting vertices, $\mathrm{frontier}(T'_k)$ contains no new conflicts. The case that no leaf of Y_b has a neighbor on level $k+1$ can be treated analogously, except that we move Y_b to become the left neighbor of Y_a in this case.

In both cases we have resolved a conflict without creating new conflicts while maintaining planarity, i.e., we reach a contradiction to the choice of π'.

Altogether, this shows that a permutation $\pi' \in \mathrm{consistent}(T_k)$ that is compatible with π and minimizes the number of conflicts is actually conflict-free, and the claim is proved. □

In particular, we have the following corollary.

Corollary 1. *Let $(G, \lhd)$ be a constrained single-source proper k-level graph and let T_j and P_j for $j \in [k]$ be the PQ-tree and the order graph of G_j, respectively. Then G admits a level-planar drawing that satisfies $\lhd$ if and only if* $\mathrm{consistent}(T_j, P_j) \neq \emptyset$ *for* $j \in [k]$.

This immediately yields a polynomial-time algorithm for CLP, which works as follows. First, compute all order graphs $P_1, \ldots, P_k$ in $O(n^5)$ time by Lemma 8. Second, start with the PQ-tree T_1, which has a single leaf. For $i \in \{2, \ldots, k\}$ compute T_i from T_{i-1} as in the algorithm of Di Battista and Nardelli, which takes $O(n)$ time over all levels, and then check whether $\mathrm{consistent}(T_i, P_i) \neq \emptyset$ using Lemma 9, which takes $O(n^2)$ time per level. The dominating part therefore is the $O(n^5)$ time resulting from the construction of the order graphs. Altogether we obtain the following theorem.

Theorem 1. *CLP can be solved in $O(n^5)$ time.*

We note that the test can be modified to output a corresponding drawing by following the construction in the proof of Lemma 11.

3.3. An efficient CLP algorithm for single-source graphs

The running time of the CLP algorithm from the previous section can be improved to $O(n + ks)$, where s is the size of $\lhd$, i.e., the number of vertex pairs that are comparable w.r.t. $\lhd$. To this end, order graphs and PQ-trees are merged into a single new data structure, a so-called constrained PQ-tree.

The main bottleneck in the algorithm from the previous section is the computation of the order graph. Consider a PQ-tree T_j of some level j that contains an inner node Y with two child edges $e = (Y, Z)$ and $f = (Y, Z')$. If the order graph P_j contains an edge (x, y) with $x \in \mathrm{yield}(\mathrm{pertinent}(Z))$ and $y \in \mathrm{yield}(\mathrm{pertinent}(Z'))$, then by Lemma 10 it also contains the edge (x', y') for each $x' \in \mathrm{yield}(\mathrm{pertinent}(Z))$ and each $y' \in \mathrm{yield}(\mathrm{pertinent}(Z'))$. On the other hand, all these edges in the order graph encode the same piece of information: e needs to occur before f in the order of children of Y. Rather than exhaustively propagating this information through the entire order graph, it is much more efficient to simply store this information directly inside the PQ-tree and to restrict the frontier to orderings that satisfy these constraints. Note that by Lemma 10, the order graph can be reconstructed from these constraints. Therefore the correctness of the approach is unaffected, however the compression of the information yields a gain of efficiency. In the following we give a more detailed description of the approach.

A *constrained PQ-tree* is a PQ-tree T that stores additional edge constraints. An *edge constraint* is an ordered pair of edges (e, f) of the tree that have the same parent node Y with the meaning that e must occur before f in the counter-clockwise order of edges around Y starting with the parent edge. See Fig. 9 for an example (constraints and parent edges are colored). This information can be stored as follows. Each edge stores a doubly linked list of all constraints it is involved in, and each constraint that involves two edges e, f also has a pointer to the corresponding entries in the constraint lists of e and f. This allows to remove a given constraint in $O(1)$ time. Since every constraint requires $O(1)$ space, a constrained PQ-tree with c constraints requires space $O(n + c)$.

The main task is to enhance the update procedure so that it correctly handles the constraints. Recall that the update step colors nodes black or white if their subtree contains only black or only white leaves. We extend this notion to edges and call an edge of the PQ-tree *black* if it is incident to a black node and white if it is incident to a white node. When updating a constrained PQ-tree, it may be necessary to replace a constraint (e, f) at a node Y by another equivalent constraint. This is the case when Y is split so that the edges corresponding to e and f end up at different parent nodes after the update; see the left (dashed, blue) and the middle (dotted, green) constraint in Fig. 9. In the following we distinguish cases based on whether e and f are terminal, black, or white.

First consider the case that e is terminal. Then (e, f) needs to be replaced by the equivalent constraint (e', f), where e' is determined as follows. Let Y be the parent node of e and f and let $e = (Y, Z)$. Then e' is the black child edge of Z that is a neighbor of e. This edge can be found in constant time. The same procedure can be used if f is terminal, e.g., the green constraint in Fig. 9. Next we consider constraints that do not involve a terminal edge. Such a constraint has to be moved if one edge is white, and the other edge is black. Assume that e is white and f is black. Then (e, f) is replaced by the constraint $((Y, x), (Y', x))$, where Y' is the split node of Y. Analogously, if e is black and f is white, (e, f) is replaced by $((Y', x), (Y, x))$. For an example, see the blue constraint in Fig. 9. Note that all replacement operations have constant running time. All other constraints do not have to be replaced, e.g., the red constraint in Fig. 9.

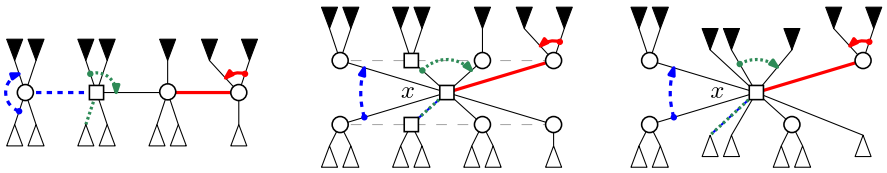

Fig. 9. The update procedure for constrained PQ-trees. Constraints are drawn as arcs, together with their respective parent edges. The dashed blue constraint involves a white and a black edge, the dotted green constraint involves a black edge and a terminal edge, and the solid red constraint involves two black edges. Observe that the update unifies the parent edges of the dashed blue and the dotted green edge.

Observe that only constraints that involve a black or terminal edge are considered for replacement. The update runs as usual, taking amortized linear time in the number of leaves that are made consecutive by Lemma 4. However, there is an additional cost for updating the constraints. We note that Q-nodes are not explicitly represented in the PQ-tree data structure and therefore references to their parents cannot necessarily be queried in constant time. However, such references *are* computed for Q-nodes on the terminal path as part of the update procedure, and only constraints around such nodes are candidates for replacement. Hence a single replacement takes $O(1)$ time. Overall the additional time for replacing constraints is linear in the number of constraints in the PQ-tree that need to be replaced, which gives an amortized running time of $O(|S| + c)$ for an update operation on a set of S leaves in a constrained PQ-tree with c constraints.

Let us now consider the concrete use constrained PQ-trees in the context of the constrained level planarity problem. The initial constrained PQ-tree T_1 for the first level is constructed as before (without any constraint). The construction of T_{j+1} from T_j is done as before, except that the considered PQ-trees now have constraints and as a post-processing, the constraints of level $j+1$ are added.

Recall that constructing T_{j+1} from T_j involves first constructing the tree $T^\star_{j+1}$ by removing obsolete leaves (without neighbors on level $j+1$), turning the remaining leaves into P-nodes, and adding one leaf for each incident edge in E_j. Clearly, this step takes time $O(|T_j| + |V_{j+1}|)$, also for constrained PQ-trees. Afterwards T_{j+1} is obtained from $T^\star_{j+1}$ by making the incoming edges of each vertex v on level $j+1$ consecutive and replacing the resulting consecutive set of edges by the single leaf v. Let $S_1, \dots, S_t$ be the sets that are made consecutive in this step. The running time for the update itself is $O(|T_j| + |V_{j+1}| + \sum_{j=1}^{t} |S_i|)$ plus for each update the cost of replacing the c constraints in the constrained PQ-tree. Here the important observation is that a constraint is replaced only if one of its edges is black or terminal. Hence, any constraint is considered for replacement at most twice during all the updates of a level, and hence the running time for computing T_{j+1} from T_j is $O(|T_j| + |V_{j+1}| + \sum_{j=1}^{t} |S_i| + c)$ for a constrained PQ-tree with c constraints. Finally, it remains to add the constraints of level $j+1$ to the new PQ-tree T_{j+1}. With a data structure for computing lowest common ancestors in $O(1)$ time [32], this can be done in $O(|T_{j+1}| + s_{j+1})$ time, where s_{j+1} is the size of $\lhd_{j+1}$, i.e., the number of vertex pairs in V_{j+1} that are comparable w.r.t. $\lhd_{j+1}$. Altogether, computing T_{j+1} from T_j therefore takes time $O(|T_j| + |V_{j+1}| + c + s_j)$ time. Since the number of constraints in the tree is at most s, it follows that $c \leq s$ and summing this over all k levels yields a running time of $O(n + k \cdot s)$ for computing the constrained PQ-tree T_j for $j \in \{1, \dots, k\}$.

To check whether a constrained PQ-tree has a frontier that is consistent with the stored constraints, we use a similar procedure to the one from Section 3.2, i.e., we check for every P-node whether the constraints around it are acyclic and for every Q-node whether the constraints around it are non-conflicting. The running time of this procedure for level j is $O(|V_j| + s)$.

In summary, to solve CLP, run the algorithm from Section 3.1 to compute the constrained PQ-tree T_j (including the level-j constraints) for each level j as shown above and then run the consistency procedure for each level. In this way, the consistent orderings of the constrained PQ-tree T_j are exactly the orderings that are consistent with both the ordinary PQ-tree of level j and the order graph of level j. It hence follows from Corollary 1 that G admits a drawing that satisfies the constraints if and only if T_j is consistent for each level $j \in [k]$. We conclude the following.

Theorem 2. *For proper single-source k-level graphs CLP can be solved in time $O(n + ks)$, where n is the number of vertices in the graph and s is the size of the constraints. In particular, PLP can be solved in $O(kn)$ time for single source proper k-level graphs.*

We note that, in general, we may assume $k \leq n$ and $s \in O(n^2)$. Therefore the running time of the algorithm is in $O(n^3)$, while for partial proper k-level graphs, we have $s \in O(n)$ and thus the running time is $O(n^2)$.

4. Complexity of the general case

It is an interesting question whether the algorithm can be extended to graphs with multiple sources. Jünger et al. [41] extend Di Battista and Nardelli's level planarity testing algorithm for multi-source graphs. To this end, they run an instance of Di Battista and Nardelli's algorithm for every source of the graph. As soon as the algorithm has progressed to a level j where multiple sources belong to the same connected component in G_j, the corresponding PQ-trees are merged. Unfortunately, the order graphs from the previous section cannot simply be reused in the multi-source case, because neither does Lemma 10 apply to multi-source graphs, nor is it obvious how constraints should be handled during the merge operation. One possible solution stems from the fact that any level graph with multiple sources can be transformed into a level graph with one source by adding one vertex and some edges, without changing its planarity. To see this, consider a planar drawing of a graph. Any level-j source can be removed by adding an edge to a suitable vertex on level $j-1$. For an example, see Fig. 10. The difficulty of this procedure is to find out *which* edges to add. If the

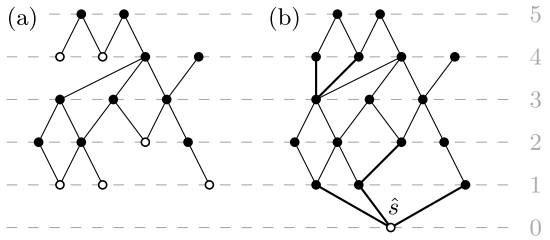

Fig. 10. Removing sources from a multi-source level graph. Part (a) shows a level graph G with six sources. Sources are highlighted as white disks, all other vertices are drawn as black vertices. By applying the procedure outlined above Corollary 2, the supergraph G' shown in part (b) is generated. This new graph has only one source. Only a single new vertex has been added, namely $\hat{s}$, and six edges, one for every former source, have been added. These new edges are drawn in bold.

input graph G has a fixed number r of sources, one can simply try all possible combinations of edges. There are at most n choices per source, leading to a total of n^r choices. For each of them we employ the algorithm from Theorem 2.

Corollary 2. *For proper k-level graphs with r sources, CLP can be solved in time $O(n^r(n + ks))$.*

Indeed, it is unlikely that order graphs can be used for multi-source graphs as well, because in this section we show that PLP is NP-complete in the general case by reducing from the NP-hard problems 3-Partition and Planar Monotone 3-Sat [22]. The first reduction shows that PLP is NP-complete even for proper level graphs whose number of levels is bounded by a constant. The second reduction shows that PLP is NP-complete even for proper level graphs with fixed combinatorial embedding and high connectivity.

4.1. 3-Partition reduction

The 3-Partition problem asks whether a given multiset of integers can be partitioned into triples (referred to as *buckets*) so that the sums of elements of all triples are equal (i.e., all buckets have the same size). More formally, an instance of 3-Partition consists of a set A of $n = 3m$ elements, a bound $B \in \mathbb{Z}^+$, and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$ such that $B/4 < s(a) < B/2$ and such that $\sum_{a\in A} s(a) = mB$. The question is then whether A can be partitioned into m disjoint sets $A_1, A_2, \dots A_m$ such that for $1 \le i \le m$ the equation $\sum_{a\in A_i} s(a) = B$ holds true. Note that each A_i must therefore contain exactly three elements from A. The 3-Partition problem is strongly NP-complete [30].

The idea of the reduction is to construct a *bucket graph*, which largely consists of gadgets of two kinds. The first kind of gadget is the *socket*. There are a total of $\sum_{a\in A} s(a)$ sockets, which are organized into m *buckets*. The second kind of gadget is the *pin*. Every element $a \in A$ is represented by a *plug* that consists of $s(a)$ pins. As the naming suggests, sockets and plugs interact with each other. In any planar embedding of the bucket graph, every plug of size $s(a)$ is *coupled* to $s(a)$ sockets, and every socket is coupled to one pin of some plug. The embedding of the sockets is fixed by the partial drawing, whereas the order of the plugs remains variable. The bucket structure ensures the required property of the 3-Partition problem that every set A_i must contain exactly three elements. Furthermore, the sets $A_1, A_2, \dots, A_m$ can be easily determined by inspecting the i-th bucket of the bucket graph.

The socket, pin and plug gadgets are shown in Fig. 11. A socket consists of a U-shaped path, together with two short y-monotone intertwined paths connected to it. The endpoints of these paths lie on the same level so that the endpoint of the path running downwards lies to the right of the endpoint of the path running upwards. Sockets are always fixed by the partial drawing. A pin is a simple serpentine-shaped path designed to fit into socket by "snaking" its way through the socket to reach the bottom. Pins are not part of the partial drawing. A pin is coupled with a socket when the end of plug p_e lies in between the vertices s_ℓ and s_r of the socket. To create adjacent sockets, take two sockets and identify the path $(\zeta, \dots, s_r)$ of the left socket and the path $(\alpha, \dots, s_\ell)$ of the right socket. To create a plug of size k, identify the roots of k pins. Fig. 11 (c) shows a plug of size two coupled with two adjacent sockets. It is readily observed that a socket cannot be coupled with more than one pin or plug without causing edge crossings.

The bucket graph is constructed step-by-step, starting out with an empty level graph. Given an instance of the 3-Partition problem, the first step is to create a total of mB pins so that the roots of all pins are on level 7. Then, for every element $a \in A$, take $s(a)$ pins and merge their roots to create one plug of size $s(a)$ and associate a with it. Given a plug p, the associated element is denoted by $a(p)$. At this point the strong NP-completeness is required because a plug of size $s(a)$ is created by merging the roots of $s(a)$ pins. Next, create mB adjacent sockets. Note that this equals the number of pins created. We will later ensure that in every drawing the plugs have to lie within the socket structure, which ensures that each pin has to be coupled with a socket and, by planarity, the pins of a plug have to be coupled with consecutive sockets. To model the partitions, we organize the sockets into *buckets* by adding $m + 1$ new vertices, called *bucket separators* on level 7 so that, for $1 \le i \le m$, the i-th bucket separator is connected to the vertex α of the $((i - 1)B + 1)$-th socket and the $(m + 1)$-th bucket separator is connected to the vertex ζ of the mB-th socket. Since the bucket separators lie on the same level as the roots of the plugs, any plug must clearly lie completely on either side of any bucket separator. Finally, to ensure that the plugs have to be embedded inside the socket structure, we add a roof construction on top of the bucket graph and attach the roots of all plugs to its center vertex as shown in the top of Fig. 12. Namely, we first create a path of length 2 whose endpoints are on level 8 and whose center point is on level 9, and we connect its left and right endpoints to the leftmost and the rightmost bucket separator, respectively, and we connect the plugs to its center vertex. To ensure that the plugs have

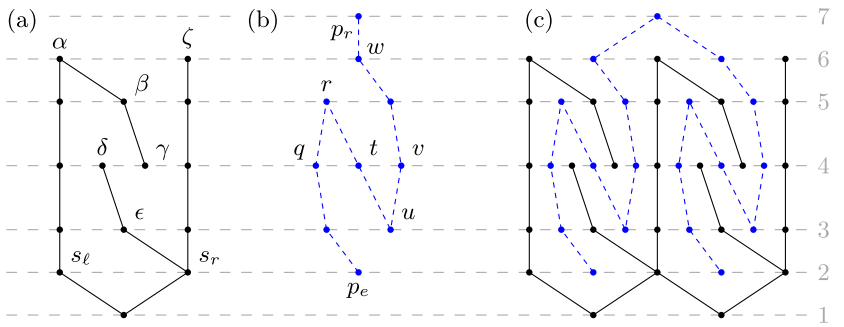

Fig. 11. Part (a) shows a socket drawn in black and part (b) shows a pin drawn dashed and blue. The pin has a root vertex p_r and an end vertex p_e. A pin is coupled with a socket when the end vertex of the pin, p_e, lies between the vertices s_ℓ and s_r of the socket. It is shown that a socket can be coupled with at most one pin. In part (c), a plug of size two is coupled with two adjacent sockets. The plug of size two is created by merging the roots of two pins. It is shown that a plug of size k is always coupled with k adjacent sockets.

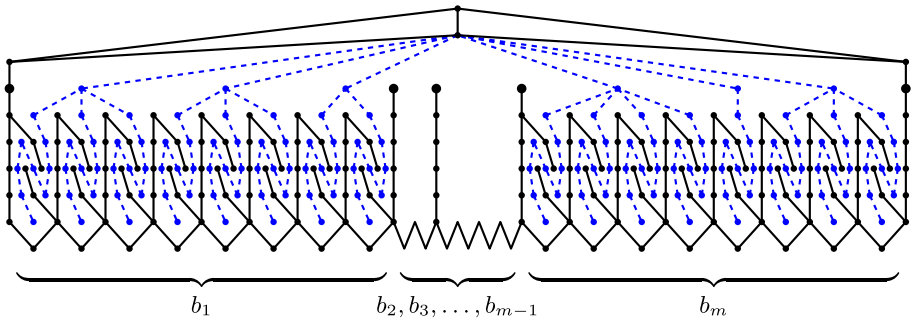

Fig. 12. A bucket graph. There are a total of m buckets, each of which consists of eight adjacent sockets, drawn in black. The bucket separators are drawn as slightly larger black vertices. The plugs are drawn dashed blue. The sockets of the first bucket are coupled with three plugs. Two of these plugs have size three and the third plug has size two. This means that one of the triples in the corresponding 3-PARTITION problem is $(3, 3, 2)$. Similarly, the m-th bucket corresponds to the triple $(4, 1, 3)$.

to be embedded in the interior of the construction, we add an additional vertex on level 10, which we connect to three vertices of the path.

At this point, the equivalence of finding a solution for a 3-PARTITION instance $\mathcal{I}$ and extending the partial drawing $\prec'$ of the corresponding bucket graph G can be shown. Assume $\mathcal{A} = \{\{a_1, a_2, a_3\}, \ldots, \{a_{3m-2}, a_{3m-1}, a_{3m}\}\}$ is a solution of the given 3-PARTITION instance $\mathcal{I}$. Since $\mathcal{A}$ is a solution for $\mathcal{I}$, it is $s(a_1) + s(a_2) + s(a_3) = B$. To complete the partial drawing $\prec'$ of G, take the first three plugs and couple them with the sockets of the first bucket. This same reasoning can be applied to all other triples in $\mathcal{A}$, resulting in a complete level-planar drawing $\prec$ of the bucket graph G. Now let G be the bucket graph corresponding to a 3-PARTITION instance $\mathcal{I}$ and assume that G has a level-planar drawing $<$. The drawing $\prec$ gives a strict total order on the roots of all plugs: $p_1 \prec p_2 \prec p_3 \prec \ldots \prec p_{3m-2} \prec p_{3m-1} \prec p_{3m}$. Consider the first bucket b_1 and recall the size restriction $B/4 < s(a) < B/2$ for each $a \in A$ of the instance $\mathcal{I}$. This means that p_1 and p_2 alone cannot fill the bucket b_1. Likewise, p_1, p_2, p_3 and p_4 cannot fit in the bucket b_1. So, the only plug candidates to fill the first bucket are the three first plugs p_1, p_2 and p_3. Each socket is coupled with a plug and because each bucket is made up of B sockets, it follows that $s(a(p_1)) + s(a(p_2)) + s(a(p_3)) \leq B$. Because no plug spans across multiple buckets, this means that $s(a(p_1)) + s(a(p_2)) + s(a(p_3)) \geq B$. Together with the previous statement, this gives $s(a(p_1)) + s(a(p_2)) + s(a(p_3)) = B$. This same reasoning can then be applied to the next three plugs p_4, p_5 and p_6, and so on, up to the last three plugs p_{3m-2}, p_{3m-1} and p_{3m}. Hence,

$$\{\{a(p_1), a(p_2), a(p_3)\}, \ldots, \{a(p_{3m-2}), a(p_{3m-1}), a(p_{3m})\}\}$$

is a solution for the 3-PARTITION instance $\mathcal{I}$. Using a similar construction we can remove all cutvertices from the bucket graph, except one; see Fig. 13. We conclude the following.

Theorem 3. *The PLP problem is NP-complete even for proper level graphs with a single cutvertex and a constant number of levels.*

We want to highlight the fact that because both the number of cutvertices and the number of levels is bounded by a constant, it is unlikely that PLP is fixed-parameter tractable with respect to either of these two parameters.

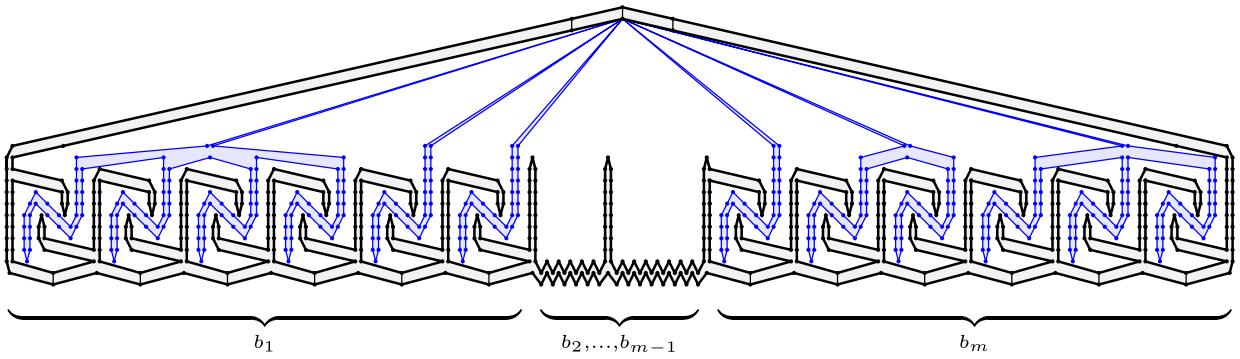

Fig. 13. A connected bucket graph with a single cutvertex. The bucket structure, consisting of two interconnected cycles (bold) is drawn in black. The plugs, each of which is a cycle, are drawn in blue. All of the plug cycles share a single vertex with the bucket structure, which is the only cutvertex in the graph.

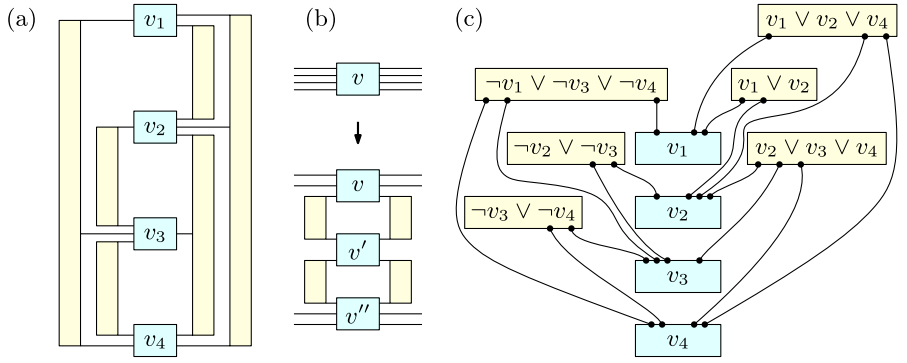

Fig. 14. A variable-clause graph (a), the construction to ensure that each literal appears at most three times (note $v \Leftrightarrow v''$) (b) and the modified version with y-monotone Jordan arcs (c).

4.2. Planar Monotone 3-Sat reduction

In this section, we present another reduction to show that PLP is NP-complete even when restricted to biconnected proper subdivisions of triconnected graphs with fixed combinatorial embedding and constant maximum vertex degree.

Recall that 3-Sat is the problem of deciding, given a set of variables V and a set of clauses C where every clause $C \in C$ contains at most three literals, whether there is a Boolean truth assignment for the variables in V so that every clause contains at least one literal that evaluates to true. A clause is called *positive* if it contains only positive literals, and *negative* if it only contains negative literals. A 3-Sat instance is *monotone* if it contains only positive and negative clauses. Further restrictions are expressed in terms of the *variable-clause graph*. The vertex set of this graph is $V \cup C$ and a variable v and a clause C are connected by an edge if and only if v or $\neg v$ occurs in C. Planar Monotone 3-Sat is the restriction of 3-Sat to monotone instances whose variable–clause graph can be drawn in such a way that all variable vertices are drawn as squares on a vertical straight line, the *line of variables*, clauses are drawn as rectangles of variable height, all edges are drawn as horizontal straight lines, and positive clauses are drawn to the right of the line of variables and negative clauses are drawn to the left of the line of variables. Fig. 14 (a) shows a variable-clause graph. De Berg and Khosravi proved that Planar Monotone 3-Sat is NP-complete [22, Theorem 1]. Note that every literal in a Planar Monotone 3-Sat instance can be assumed to occur in at most three clauses. See Fig. 14 (b) to see how for a variable v whose literals appear more than three times new variables v', v'' and clauses $(v \vee v'), (\neg v \vee \neg v'), (v' \vee v''), (\neg v' \vee \neg v'')$ can be created. Note $v \Leftrightarrow v''$, so some literals of v can be replaced by literals of v''. This construction can be repeated until each literal appears in at most three clauses.

To transform a Planar Monotone 3-Sat instance $\mathcal{I}$ into a PLP instance $(G, \prec')$, parts of the variable clause graph are replaced by level graph gadgets together with partial drawings thereof. First, the drawing is altered so that all edges are drawn as y-monotone Jordan arcs that connect to the bottom of clause rectangles and to the top of variable rectangles; see Fig. 14 (c). Then, every variable is replaced by a *variable gadget*, every clause is replaced by a *clause gadget* and every edge is replaced by a *pipe gadget*.

The pipe gadget consists of two channels that are fixed by the partial drawing, and one *conductor* that remains to be drawn. Fig. 15 shows a pipe gadget, where the channels are drawn in black and the conductor is drawn in blue. Depending on the completion of the partial drawing, the conductor may flow through either the left or the right channel. Note the construction in the middle, which serves to connect the inner part separating the two channels to the boundary. It splits the conductor into a lower and an upper part. The partial drawing is used to force the upper end of the lower part of the conductor between the claw-like structure at the lower end of the upper part of the conductor. This means that even though the conductor is split into two parts, both of them must lie in the same channel. Each pipe gadget is then merged with a clause gadget at the top and with a variable gadget at the bottom. The

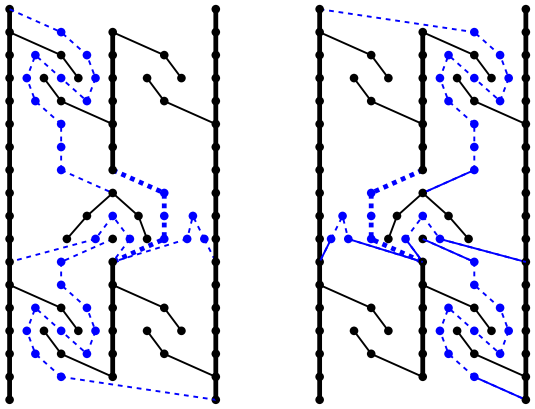

Fig. 15. The two states of a pipe gadget. The partial drawing is solid black, the conductor is dashed blue. Note the construction in the middle, which serves to connect the inner part separating the two channels to the boundary.

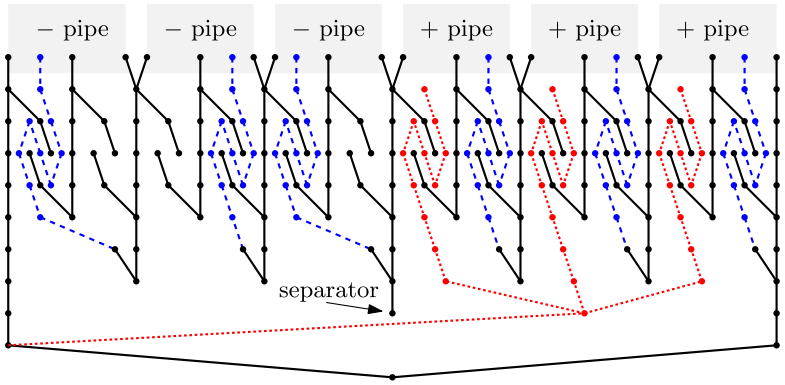

Fig. 16. A variable gadget configured as false. The partial drawing is solid black, the conductors are dashed blue and the pin structure of the variable is dashed red. The pin structure must lie entirely left or right of the separator vertex ensures that the pin structure lies entirely left or right of the separator.

negative clauses, which lie left of the variables, are connected to the leftmost three pipes that leave a variable, while the positive clauses, which lie right of the variables, are connected to the rightmost three pipes. This way, information is relayed between clause gadgets and variable gadgets.

The variable gadget consists of six merged pipe ends, separated in the middle by a separator vertex, and three pins merged at their lower endpoint. This pin structure must lie either completely to the left of the separator, or completely to its right. Depending on this, the variable is configured as true or false. Fig. 16 shows a variable gadget configured as false.

Property 1. *Every planar drawing of the variable gadget that extends the partial drawing has the following property. If it is configured as false (as true), the conductors of all pipes leading to the gadgets of positive (negative) clauses all run through their right channels. The channel choice of the conductors of all pipes leading to negative (positive) clauses is not restricted.*

The clause gadget consists of three merged pipe ends where the top vertex of the left boundary of the left pipe and the top vertex of the right boundary of the right pipe are further connected by a path of length 2. Moreover, a pin is attached to the left endpoints of the path; see Fig. 17, which shows a clause gadget where the first literal is satisfied. In a drawing of the clause that extends the given partial drawing, the pin must be drawn inside one of the pipes, thereby forcing the conductor of that pipe to flow through the left channel.

Property 2. *In a drawing of the clause gadget that extends the given partial drawing, the conductor of at least one of the three connected pipes must run through the left channel.*

Given an instance $\mathcal{I}$ of Planar Monotone 3-Sat, the corresponding PLP instance $(G, \prec')$ can clearly be computed in polynomial time. Assume that there exists a planar completion of $\prec'$. Consider a positive clause C. Property 2 states that the conductor of at least one of the connected pipes must run through the left channel. Suppose that the variable X at the other end of this pipe is configured as false. Then Property 1 states that the conductors of all pipes leading to positive clauses, including C, must run through the right

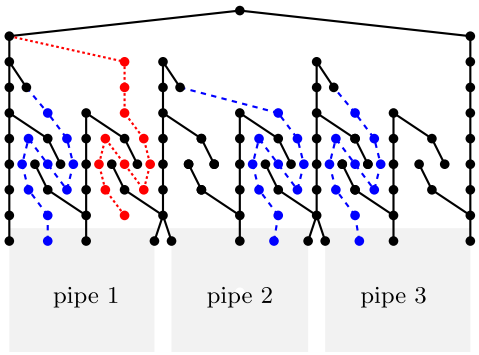

Fig. 17. The clause gadget; pipes 1 and 3 transmit a literal value that satisfies the clause. The conductors are dashed blue, the pin of the clause is dotted red.

channel. This is a contradiction, so X must be configured as true and C is satisfied. A similar argument can be made when C is a negative clause.

Conversely, suppose that all variable gadgets are configured according to a satisfying assignment. Consider a positive clause C. Since C is satisfied, at least one of its variables X is true. Let the conductor of the pipe P connecting C and X flow through the left channel. This is possible because according to Property 1, only the conductors of pipes leading to negative clauses must run through the right channel. A similar argument can be made when C is a negative clause. Any remaining conductors may flow through either channel.

Hence, a planar completion of $\prec'$ exists if and only if there is a satisfying variable assignment for $\mathcal{I}$. With the NP-completeness of Planar Monotone 3-Sat, this gives the following.

Theorem 4. *The problem PLP is NP-complete even for connected proper level graphs.*

The reduction can be strengthened by replacing each vertex and each edge of the gadgets by a suitable internally triangulated graph (see Fig. 18) such that the resulting graph is a subdivision of a triconnected graph. Then the combinatorial embedding of the graph is fixed. Moreover, the maximum vertex degree is bounded, and there is only a constant number of sources per level. Note that here it is crucial that the graph we start with is connected.

Theorem 5. *The PLP problem remains NP-complete when restricted to proper subdivisions of triconnected graphs of bounded degree.*

Because triconnected planar graphs have a fixed combinatorial embedding this result also implies NP-completeness of the PLP problem for level graphs with a fixed combinatorial embedding.

5. Conclusion

In this paper we studied a constrained version of level planarity that allows to specify constraints on the linear order of vertices in the sought drawing. This problem contains as a special case the problem of extending a partial drawing in the level-planar setting, a problem that has recently received considerable interest for many other drawing styles and also other graph representations. Our algorithm for single-source proper k-level graphs takes $O(n + ks)$ time, where n is the number of vertices and s is the size of the constraints. For non-proper single-source graphs, we first subdivide them to make the graph proper, which increases the graph size to $O(nk)$ and therefore results in a running time of $O(nk + ks)$. It is an interesting question whether the running time for non-proper graphs can be improved.

Another question is whether the restriction to instances with a single source can be alleviated. While our strong hardness results leave little room for polynomial-time algorithms for much larger classes of instances than single-source graphs, in view of Corollary 2, our most important open question is whether CLP is fixed-parameter tractable with respect to the number of sources in the graph. The recent algorithm for showing that upward planarity is FPT with respect to the number of sources [18] as well as recently developed SPQR-tree variant for single-source level planarity [17] may be interesting starting points in this regard.

Another interesting question is whether our techniques can be adapted for related drawing styles, in particular radial drawings, where levels are represented by concentric circles rather than horizontal lines, and upward drawings, where the levels of vertices are not fixed.

CRediT authorship contribution statement

Guido Brückner: Writing – review & editing, Writing – original draft, Visualization, Methodology, Investigation, Formal analysis, Conceptualization. **Ignaz Rutter:** Writing – review & editing, Writing – original draft, Supervision, Project administration, Methodology, Formal analysis, Conceptualization.

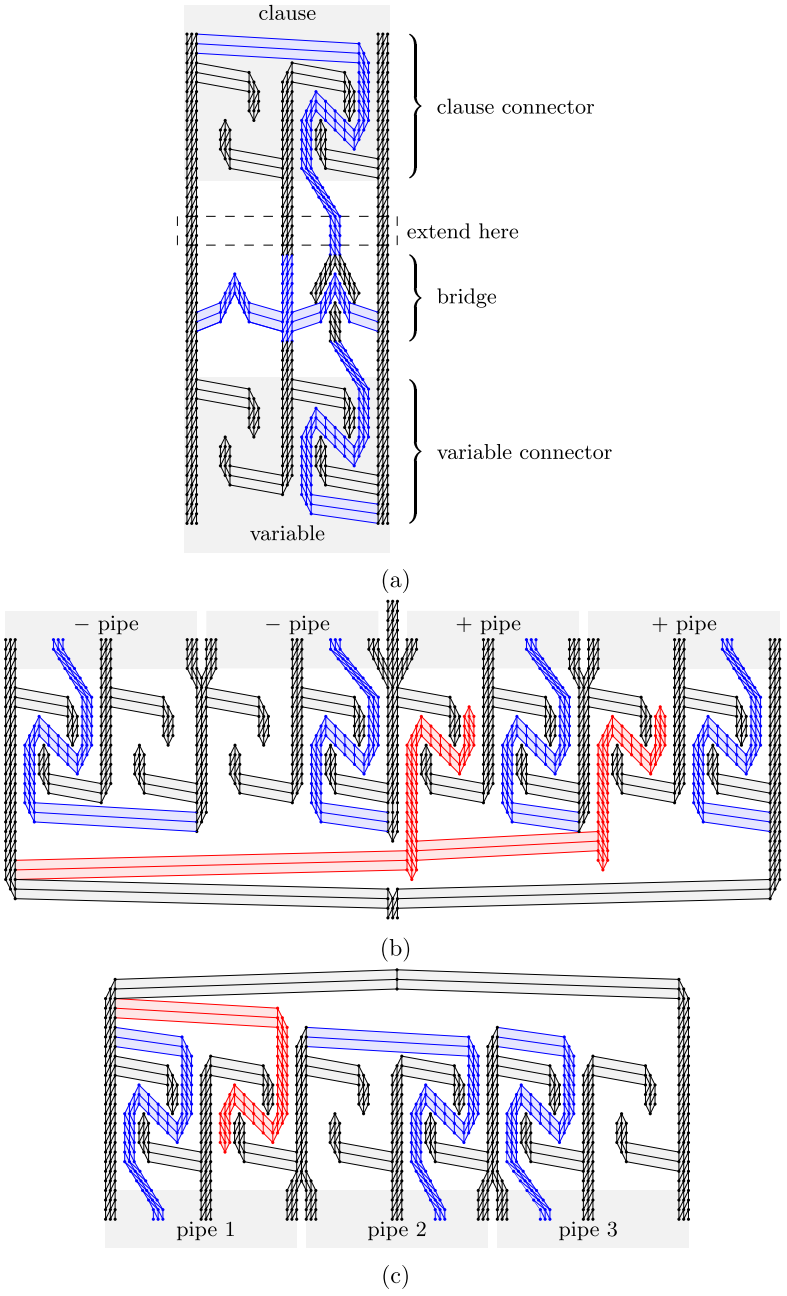

Fig. 18. Augmented gadget pipe gadget (a), variable gadget (b) and clause gadget(c), where the graph forms a subdivision of a triconnected graph. Due to space reasons one positive pipe and one negative pipe of the variable have been omitted compared to Fig. 16.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We thank Giordano Da Lozzo for the idea behind the plug/socket gadget.

Data availability

No data was used for the research described in the article.

References

[1] Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, Ignaz Rutter, Intersection-link representations of graphs, J. Graph Algorithms Appl. 21 (4) (2017) 731–755.
[2] Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, Ignaz Rutter, Testing cyclic level and simultaneous level planarity, Theor. Comput. Sci. 804 (2020) 161–170.
[3] Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Vincenzo Roselli, The importance of being proper: (in clustered-level planarity and T-level planarity), Theor. Comput. Sci. 571 (2015) 1–9.
[4] Patrizio Angelini, Giordano Da Lozzo, Daniel Neuwirth, Advancements on SEFE and partitioned book embedding problems, Theor. Comput. Sci. 575 (2015) 71–89.
[5] Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, Vít Jelínek, Jan Kratochvíl, Maurizio Patrignani, Ignaz Rutter, Testing planarity of partially embedded graphs, ACM Trans. Algorithms 11 (4) (2015) 32.
[6] Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, Ignaz Rutter, Testing the simultaneous embeddability of two graphs whose intersection is a biconnected or a connected graph, J. Discret. Algorithms 14 (2012) 150–172.
[7] Christian Bachmaier, Franz-Josef Brandenburg, Michael Forster, Radial level planarity testing and embedding in linear time, J. Graph Algorithms Appl. 9 (1) (2005) 53–97.
[8] Thomas Bläsius, Simon D. Fink, Ignaz Rutter, Synchronized planarity with applications to constrained planarity problems, in: Petra Mutzel, Rasmus Pagh, Grzegorz Herman (Eds.), 29th Annual European Symposium on Algorithms, ESA 2021, September 6–8, 2021, Lisbon, Portugal (Virtual Conference), in: LIPIcs, vol. 204, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 19.
[9] Thomas Bläsius, Annette Karrer, Ignaz Rutter, Simultaneous embedding: edge orderings, relative positions, cutvertices, Algorithmica 80 (4) (2018) 1214–1277.
[10] Thomas Bläsius, Stephen G. Kobourov, Ignaz Rutter, Simultaneous embedding of planar graphs, in: Roberto Tamassia (Ed.), Handbook on Graph Drawing and Visualization, Chapman and Hall/CRC, 2013, pp. 349–381.
[11] Thomas Bläsius, Ignaz Rutter, Disconnectivity and relative positions in simultaneous embeddings, Comput. Geom. 48 (6) (2015) 459–478.
[12] Thomas Bläsius, Ignaz Rutter, Simultaneous pq-ordering with applications to constrained embedding problems, ACM Trans. Algorithms 12 (2) (2016) 16.
[13] Kellogg S. Booth, George S. Lueker, Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms, J. Comput. Syst. Sci. 13 (3) (1976) 335–379.
[14] Franz-Josef Brandenburg, Upward planar drawings on the standing and the rolling cylinders, Comput. Geom. 47 (1) (2014) 25–41.
[15] Guido Brückner, Markus Himmel, Ignaz Rutter, An SPQR-tree-like embedding representation for upward planarity, in: Archambault Daniel, Csaba D. Tóth (Eds.), Graph Drawing and Network Visualization - 27th International Symposium, Proceedings, GD 2019, Prague, Czech Republic, September 17–20, 2019, in: Lecture Notes in Computer Science, vol. 11904, Springer, 2019, pp. 517–531.
[16] Guido Brückner, Ignaz Rutter, Peter Stumpf, Level planarity: transitivity vs. even crossings, in: Therese C. Biedl, Andreas Kerren (Eds.), Graph Drawing and Network Visualization - 26th International Symposium, Proceedings, GD 2018, Barcelona, Spain, September 26-28, 2018, in: Lecture Notes in Computer Science, vol. 11282, Springer, 2018, pp. 39–52.
[17] Guido Brückner, Ignaz Rutter, An spqr-tree-like embedding representation for level planarity, J. Comput. Geom. 14 (1) (2023).
[18] Steven Chaplick, Emilio Di Giacomo, Fabrizio Frati, Robert Ganian, Chrysanthi N. Raftopolou, Kirill Simonov, Parameterized algorithms for upward planarity, in: Proceedings of the 38th International Symposium on Computational Geometry (SoCG'22), in: LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 26.
[19] Steven Chaplick, Paul Dorbec, Jan Kratochvíl, Mickaël Montassier, Juraj Stacho, Contact representations of planar graphs: extending a partial representation is hard, in: Dieter Kratsch, Ioan Todinca (Eds.), Graph-Theoretic Concepts in Computer Science - 40th International Workshop, Revised Selected Papers, WG 2014, Nouan-le-Fuzelier, France, June 25–27, 2014, in: Lecture Notes in Computer Science, vol. 8747, Springer, 2014, pp. 139–151.
[20] Steven Chaplick, Radoslav Fulek, Pavel Klavík, Extending partial representations of circle graphs, J. Graph Theory 91 (4) (2019) 365–394.
[21] Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Extending upward planar graph drawings, Comput. Geom. 91 (2020) 101668.
[22] Mark de Berg, Amirali Khosravi, Optimal binary space partitions for segments in the plane, Int. J. Comput. Geom. Appl. 22 (3) (2012) 187–206.
[23] Giuseppe Di Battista, Enrico Nardelli, Hierarchies and planarity theory, IEEE Trans. Syst. Man Cybern. 18 (6) (1988) 1035–1046.
[24] Alejandro Estrella-Balderrama, Elisabeth Gassner, Michael Jünger, Merijam Percan, Marcus Schaefer, Michael Schulz, Simultaneous geometric graph embeddings, in: Seok-Hee Hong, Takao Nishizeki, Wu Quan (Eds.), Graph Drawing, 15th International Symposium, Revised Papers, GD 2007, Sydney, Australia, September 24–26, 2007, in: Lecture Notes in Computer Science, vol. 4875, Springer, 2007, pp. 280–290.
[25] Simon D. Fink, Matthias Pfretzschner, Ignaz Rutter, Experimental comparison of pc-trees and pq-trees, ACM J. Exp. Algorithmics 28 (2023) 1.10.
[26] Simon D. Fink, Matthias Pfretzschner, Ignaz Rutter, Peter Stumpf, Level planarity is more difficult than we thought (poster abstract), in: Stefan Felsner, Karsten Klein (Eds.), Proceedings of the 32nd International Symposium on Graph Drawing and Network Visualization (GD'24), in: LIPIcs, vol. 320, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 50.
[27] Radoslav Fulek, Michael J. Pelsmajer, Marcus Schaefer, Hanani-Tutte for radial planarity, J. Graph Algorithms Appl. 21 (1) (2017) 135–154.
[28] Radoslav Fulek, Michael J. Pelsmajer, Marcus Schaefer, Daniel Štefankovič, Thirty Essays on Geometric Graph Theory, chapter Hanani-Tutte, Monotone Drawings, and Level-Planarity, Springer, 2013.
[29] Radoslav Fulek, Csaba D. Tóth, Atomic embeddability, clustered planarity, and thickenability, in: Shuchi Chawla (Ed.), Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, SIAM, 2020, pp. 2876–2895.
[30] M.R. Garey, David S. Johnson, Complexity results for multiprocessor scheduling under resource constraints, SIAM J. Comput. 4 (4) (1975) 397–411.
[31] Luca Grilli, Seok-Hee Hong, Jan Kratochvíl, Ignaz Rutter, Drawing simultaneously embedded graphs with few bends, in: Christian A. Duncan, Antonios Symvonis (Eds.), Graph Drawing - 22nd International Symposium, Revised Selected Papers, GD 2014, Würzburg, Germany, September 24–26, 2014, in: Lecture Notes in Computer Science, vol. 8871, Springer, 2014, pp. 40–51.
[32] Dov Harel, Robert E. Tarjan, Fast algorithms for finding nearest common ancestors, SIAM J. Comput. 13 (2) (1984) 338–355.
[33] Martin Harrigan, Patrick Healy, Practical level planarity testing and layout with embedding constraints, in: Seok-Hee Hong, Takao Nishizeki, Wu Quan (Eds.), Graph Drawing, 15th International Symposium, Revised Papers, GD 2007, Sydney, Australia, September 24–26, 2007, in: Lecture Notes in Computer Science, vol. 4875, Springer, 2007, pp. 62–68.
[34] Patrick Healy, Nikola S. Nikolov, Hierarchical drawing algorithms, in: Roberto Tamassia (Ed.), Handbook on Graph Drawing and Visualization, Chapman and Hall/CRC, 2013, pp. 409–453.
[35] Lenwood S. Heath, Sriram V. Pemmaraju, Recognizing leveled-planar dags in linear time, in: Proc. Graph Drawing (GD'95), in: Lecture Notes in Computer Science, vol. 1027, Springer Verlag, 1996, pp. 300–311.
[36] Wen-Lian Hsu, Ross M. McConnell, PC trees and circular-ones arrangements, Theor. Comput. Sci. 296 (1) (2003) 99–116.

[37] Krishnam Raju Jampani, Anna Lubiw, Simultaneous interval graphs, in: Otfried Cheong, Kyung-Yong Chwa, Kunsoo Park (Eds.), Algorithms and Computation - 21st International Symposium, Proceedings, Part I, ISAAC 2010, Jeju Island, Korea, December 15–17, 2010, in: Lecture Notes in Computer Science, vol. 6506, Springer, 2010, pp. 206–217.
[38] Krishnam Raju Jampani, Anna Lubiw, The simultaneous representation problem for chordal, comparability and permutation graphs, J. Graph Algorithms Appl. 16 (2) (2012) 283–315.
[39] Vít Jelínek, Jan Kratochvíl, Ignaz Rutter, A Kuratowski-type theorem for planarity of partially embedded graphs, Comput. Geom. 46 (4) (2013) 466–492.
[40] Michael Jünger, Sebastian Leipert, Level planar embedding in linear time, J. Graph Algorithms Appl. 6 (1) (2002) 67–113.
[41] Michael Jünger, Sebastian Leipert, Petra Mutzel, Level planarity testing in linear time, in: Sue Whitesides (Ed.), Graph Drawing, 6th International Symposium, Proceedings, GD'98, Montréal, Canada, August 1998, in: Lecture Notes in Computer Science, vol. 1547, Springer, 1998, pp. 224–237.
[42] Michael Jünger, Sebastian Leipert, Petra Mutzel, Pitfalls of using pq-trees in automatic graph drawing, in: Di Battista (Ed.), Proc. Graph Drawing (GD'97), in: Lecture Notes in Computer Science, vol. 1353, Springer Verlag, 1997, pp. 193–204.
[43] Pavel Klavík, Jan Kratochvíl, Tomasz Krawczyk, Bartosz Walczak, Extending partial representations of function graphs and permutation graphs, in: Leah Epstein, Paolo Ferragina (Eds.), Algorithms - ESA 2012 - 20th Annual European Symposium, Proceedings, Ljubljana, Slovenia, September 10–12, 2012, in: Lecture Notes in Computer Science, vol. 7501, Springer, 2012, pp. 671–682.
[44] Pavel Klavík, Jan Kratochvíl, Yota Otachi, Ignaz Rutter, Toshiki Saitoh, Maria Saumell, Tomás Vyskocil, Extending partial representations of proper and unit interval graphs, Algorithmica 77 (4) (2017) 1071–1104.
[45] Pavel Klavík, Jan Kratochvíl, Yota Otachi, Toshiki Saitoh, Extending partial representations of subclasses of chordal graphs, Theor. Comput. Sci. 576 (2015) 85–101.
[46] Pavel Klavík, Jan Kratochvíl, Yota Otachi, Toshiki Saitoh, Tomás Vyskocil, Extending partial representations of interval graphs, Algorithmica 78 (3) (2017) 945–967.
[47] Boris Klemz, Günter Rote, Ordered level planarity and its relationship to geodesic planarity, bi-monotonicity, and variations of level planarity, ACM Trans. Algorithms 15 (4) (2019) 53.
[48] Tomasz Krawczyk, Bartosz Walczak, Extending partial representations of trapezoid graphs, in: Hans L. Bodlaender, Gerhard J. Woeginger (Eds.), Graph-Theoretic Concepts in Computer Science - 43rd International Workshop, Revised Selected Papers, WG 2017, Eindhoven, the Netherlands, June 21–23, 2017, in: Lecture Notes in Computer Science, vol. 10520, Springer, 2017, pp. 358–371.
[49] Sebastian Leipert, Level Planarity Testing and Embedding in Linear Time, PhD thesis, Universität Köln, 1998.
[50] Tamara Mchedlidze, Martin Nöllenburg, Ignaz Rutter, Extending convex partial drawings of graphs, Algorithmica 76 (1) (2016) 47–67.
[51] Maurizio Patrignani, On extending a partial straight-line drawing, Int. J. Found. Comput. Sci. 17 (5) (2006) 1061–1070.
[52] Helen C. Purchase, Robert F. Cohen, Murray I. James, An experimental study of the basis for graph drawing algorithms, ACM J. Exp. Algorithmics 2 (4) (1997).
[53] Bert Randerath, Ewald Speckenmeyer, Endre Boros, Peter L. Hammer, Alexander Kogan, Kazuhisa Makino, Bruno Simeone, Ondrej Cepek, A satisfiability formulation of problems on level graphs, Electron. Notes Discrete Math. 9 (2001) 269–277.
[54] Ignaz Rutter, Simultaneous embedding, in: Seok-Hee Hong, Takeshi Tokuyama (Eds.), Beyond Planar Graphs, Communications of NII Shonan Meetings, Springer, 2020, pp. 237–265.
[55] Marcus Schaefer, Toward a theory of planarity: Hanani-Tutte and planarity variants, J. Graph Algorithms Appl. 17 (4) (2013) 367–440.
[56] Wei-Kuan Shih, Wen-Lian Hsu, A new planarity test, Theor. Comput. Sci. 223 (1–2) (1999) 179–191.
[57] Kozo Sugiyama, Shojiro Tagawa, Mitsuhiko Toda, Methods for visual understanding of hierarchical system structures, IEEE Trans. Syst. Man Cybern. 11 (2) (1981) 109–125.
[58] Stephen Warshall, A theorem on Boolean matrices, J. ACM 9 (1) (1962) 11–12.