Constrained Boundary Labeling

Thomas Depian
Algorithms and Complexity Group, TU Wien, Austria

Martin Nöllenburg
Algorithms and Complexity Group, TU Wien, Austria

Soeren Terziadis
Algorithms cluster, TU Eindhoven, The Netherlands

Markus Wallinger
Chair for Efficient Algorithms, Technical University of Munich, Germany

Abstract

Boundary labeling is a technique in computational geometry used to label dense sets of feature points in an illustration. It involves placing labels along an axis-aligned bounding box and connecting each label with its corresponding feature point using non-crossing leader lines. Although boundary labeling is well-studied, semantic constraints on the labels have not been investigated thoroughly. In this paper, we introduce *grouping* and *ordering constraints* in boundary labeling: Grouping constraints enforce that all labels in a group are placed consecutively on the boundary, and ordering constraints enforce a partial order over the labels. We show that it is NP-hard to find a labeling for arbitrarily sized labels with unrestricted positions along one side of the boundary. However, we obtain polynomial-time algorithms if we restrict this problem either to uniform-height labels or to a finite set of candidate positions. Finally, we show that finding a labeling on two opposite sides of the boundary is NP-complete, even for uniform-height labels and finite label positions.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Computational geometry; Theory of computation → Problems, reductions and completeness; Human-centered computing → Geographic visualization

Keywords and phrases Boundary labeling, Grouping constraints, Ordering constraints

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.26

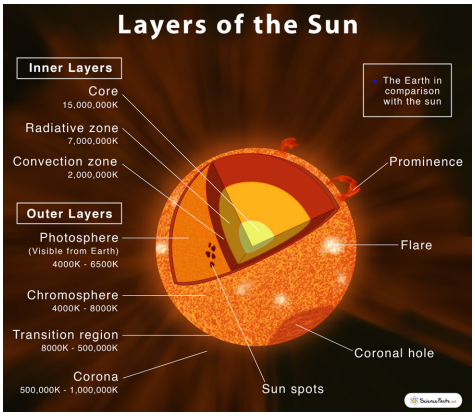
Related Version *Full Version*: `https://arxiv.org/abs/2402.12245` [10]

Funding *Thomas Depian*: Vienna Science and Technology Fund (WWTF) [10.47379/ICT22029].
Martin Nöllenburg: Vienna Science and Technology Fund (WWTF) [10.47379/ICT19035].
Soeren Terziadis: Vienna Science and Technology Fund (WWTF) [10.47379/ICT19035] and European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 101034253.
Markus Wallinger: Vienna Science and Technology Fund (WWTF) [10.47379/ICT19035].

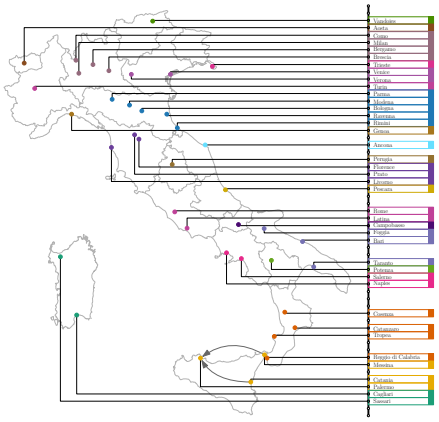
1 Introduction

Annotating features of interest with textual information in illustrations, e.g., in technical, medical, or geographic domains, is an important and challenging task in graphic design and information visualization. One common guideline when creating such labeled illustrations is to "not obscure important details with labels" [9, p. 35]. Therefore, for complex illustrations, designers tend to place the labels outside the illustrations, creating an *external labeling* as shown in Figure 1a. Feature points, called *sites*, are connected to descriptive labels with non-crossing polyline *leaders*, while optimizing an objective function, e.g., the leader length.

External labeling is a well-studied area both from a practical visualization perspective and from a formal algorithmic perspective [5]. One aspect of external labeling that has not yet been thoroughly studied in the literature, though, and which we investigate in this paper,



(a) Schematic of the sun. © ScienceFacts.net [7]; reproduced with permission.



(b) Cities in Italy. Labeling with *po*-leaders created by our algorithm described in Section 2.2.

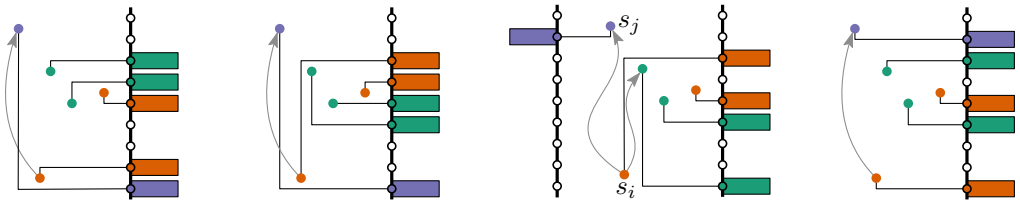
Figure 1 Labelings that adhere to semantic constraints.

is that of constraining the placement of (subsets of) labels in the optimization process. The sequential arrangement of external labels along the boundary of the illustration creates new spatial proximities between the labels that do not necessarily correspond to the geometric proximity patterns of the sites in the illustration. Hence, it is of interest in many applications to put constraints on the grouping and ordering of these labels in order to improve the readability and semantic coherence of the labeled illustration. Examples of such constraints could be to group labels of semantically related sites or to restrict the top-to-bottom order of certain labels to reflect some ordering of their sites in the illustration; see Figure 1a, where the inner and outer layers of the sun are grouped and ordered from the core to the surface.

More precisely, we study such constrained labelings in the *boundary labeling* model, which is a well-studied special case of external labelings. Here, the labels must be placed along a rectangular boundary around the illustration [4]. Initial work placed the labels on one or two sides of the boundary, usually the left and right sides. For uniform-height labels, polynomial-time algorithms to compute a labeling that minimizes the length of the leaders [4] or more general optimization functions [6] have been proposed. Polynomial-time approaches to compute a labeling with equal-sized labels on (up to) all four sides of the boundary are also known [21]. For non-uniform height labels, NP-hardness has been shown in the general two-sided [4], and in different one-sided settings [3, 12]. Several leader styles have been considered, and we refer to the book of Bekos et al. [5] and the user study of Barth et al. [1] for an overview. In this paper, we will focus on a frequently used class of L-shaped leaders, called *po-leaders*, that consist of two segments: one is ***p***arallel and the other ***o***rthogonal to the side of the boundary on which the label is placed [4], see Figure 1b. These *po*-leaders turned out as the recommended leader type in the study of Barth et al. [1] as they performed well in various readability tasks and received high user preference ratings.

The literature considered various extensions of boundary labeling [2, 12, 17, 18], and we broaden this body of work with our paper that aims at systematically investigating the above-mentioned constraints in boundary labeling from an algorithmic perspective.

Problem Description. In the following, we use the taxonomy of Bekos et al. [5] where applicable. Let $\mathcal{S}$ be a set of n sites in $\mathbb{R}^2$ enclosed in an axis-parallel bounding box $\mathcal{B}$ and in general position, i.e., no two sites share the same x- or y-coordinate. For each site $s_i \in \mathcal{S}$, we



(a) Length-minimal. **(b)** Bend-minimal. **(c)** A 2-sided labeling. **(d)** Non-admissible.

Figure 2 Colors indicate grouping and arrows ordering constraints. Labelings that are optimal with respect to **(a)** the total leader length and **(b)** the number of bends. In the 2-sided layout **(c)** the ordering constraint $s_i \preccurlyeq s_j$ is not enforced since ℓ_i and ℓ_j are on different sides. Note that **(d)** is a planar but non-admissible length-minimal labeling.

have an open rectangle ℓ_i of height $h(\ell_i)$ and some width, which we call the *label* of the site. The rectangles model the (textual) labels, which are usually a single line of text in a fixed font size, as their bounding boxes. Hence, we often restrict ourselves to uniform-height labels, but neglect their width. The *po*-leader $\lambda_i = (s_i, c_i)$ is a polyline consisting of (up to) one vertical and one horizontal segment and connects site s_i with the *reference point* c_i, which is a point on the boundary $\mathcal{B}$ at which we attach the label ℓ_i. The point where λ_i touches ℓ_i is called the *port* of ℓ_i. We define the port for each label ℓ to be at half its height, i.e., we use *fixed* ports. Hence, the position of c_i uniquely determines the position of the port p_i and thus the placement of the label ℓ_i. We denote with Λ the set of all possible leaders and with $\mathcal{C}$ the set of all possible reference points. In a b-sided boundary labeling $\mathcal{L}\colon \mathcal{S} \to \mathcal{C}$, we route for each site $s \in \mathcal{S}$ a leader λ to the reference point $\mathcal{L}(s)$ on the right ($b = 1$) or the right and left ($b = 2$) side of $\mathcal{B}$, such that we can (in a post-processing step) place the label ℓ for s at the reference point $\mathcal{L}(s)$ and no two labels will overlap. If $\mathcal{C}$ consists of a finite set of m candidate reference points on $\mathcal{B}$, we say that we have *fixed (candidate) reference points*, otherwise $\mathcal{C}$ consists of the respective side(s) of $\mathcal{B}$, which is called *sliding (candidate) reference points*. In the following, we call the reference points in $\mathcal{C}$ simply *candidates*. A labeling is called *planar* if no two labels overlap and there is no leader-leader or leader-site crossing. We can access the x- and y-coordinate of a site, port, or candidate with $x(\cdot)$ and $y(\cdot)$.

In our constrained boundary labeling setting, we are given a tuple of constraints $(\Gamma, \preccurlyeq)$ consisting of a family of k grouping constraints Γ and a partial order $\preccurlyeq$ on the sites. A *grouping* constraint $\emptyset \neq \mathcal{G} \subseteq \mathcal{S}$ enforces that the labels for the sites in $\mathcal{G}$ appear consecutively on the same side of the boundary, as in Figures 2a–2c, but in general there can be gaps between two labels of the same group; compare Figures 2a and 2b. An *ordering* constraint $s_i \preccurlyeq s_j$ enforces for the labeling $\mathcal{L}$ to have $y(\mathcal{L}(s_i)) \geq y(\mathcal{L}(s_j))$ but only if s_i and s_j are labeled on the same side of $\mathcal{B}$. Hence, if the labels ℓ_i and ℓ_j are placed on the same side of $\mathcal{B}$, then the ordering constraint enforces that ℓ_j must not appear above ℓ_i. On the other hand, if ℓ_i and ℓ_j are on different sides of $\mathcal{B}$, then the constraint $s_i \preccurlyeq s_j$ has no effect; see also Figure 2c. This interpretation is motivated by the Gestalt principle of proximity [30], as the spatial distance between labels on opposite sides is usually large and we thus perceive labels on the same side as belonging together. Furthermore, this interpretation of ordering constraints has already been applied in hand-made labelings [14]. Note that in general $\preccurlyeq$ may contain reflexive constraints, which will be fulfilled by any labeling and can thus be removed. The one-sided model in addition implicitly fulfills any transitive constraint. Hence, we work in the one-sided model with the transitive reduction of $(\mathcal{S}, \preccurlyeq)$. In the following, we denote with r the number of ordering constraints in the respective model.

A labeling *respects* the grouping/ordering constraints if all the grouping/ordering constraints are satisfied. Furthermore, the grouping/ordering constraints are *consistent* if there exists a (not necessarily planar) labeling that respects them. Similarly, the constraints $(\Gamma, \preccurlyeq)$ are consistent if there exists a labeling that respects Γ and $\preccurlyeq$ simultaneously. Finally, a labeling is *admissible* if it is planar and respects the constraints. If an admissible labeling exists, we want to optimize a quality criterion expressed by a function $f\colon \Lambda \to \mathbb{R}_0^+$. In this paper, f measures the length or the number of bends of a leader, which are the most commonly used criteria [5]. Figure 2 highlights the differences and shows in (d) that an optimal admissible labeling might be, with respect to f, worse than its planar (non-admissible) counterpart.

In an instance $\mathcal{I}$ of the Constrained b-Sided Boundary Labeling problem (b-CBL in short), we want to find an admissible b-sided *po*-labeling $\mathcal{L}^*$ for $\mathcal{I}$ (possibly on a set of m candidates $\mathcal{C}$) that minimizes $\sum_{s\in\mathcal{S}} f((s, \mathcal{L}^*(s))$ or report that no admissible labeling exists.

Related Work on Constrained External Labeling. Our work is in line with (recent) efforts to integrate semantic constraints into the external labeling model. The survey of Bekos et al. [5] reports papers that group labels. Some results consider heuristic label placements in interactive 3D visualizations [19, 20] or group (spatially close) sites together to label them with a single label [11, 24, 28], bundle the leaders [25], or align the labels [29]. These papers are usually targeted at applications and do not aim for exact algorithms or formal complexity bounds. Moreover, the grouping is often not part of the input but rather determined by clustering similar sites. To the best of our knowledge, Niedermann et al. [26] are the first that support the explicit grouping of labels while ensuring a planar labeling. They proposed a contour labeling algorithm, a generalization of boundary labeling, that can be extended to group sets of labels as hard constraints in their model. However, they did not analyze this extension in detail and do not support ordering constraints. Recently, Gedicke et al. [16] tried to maximize the number of respected groups that arise from the spatial proximity of the sites or their semantics, i.e., they reward a labeling also based on the number of consecutive labels from the same group. They disallow assigning a site to more than one group, but see combining spatial and semantic groups as an interesting direction for further research. We work towards that goal, as we allow grouping constraints to overlap. Finally, Klawitter et al. [23] visualized geophylogenies by embedding a binary (phylogenetic) tree on one side of the boundary. Each leaf of the tree corresponds to a site, and we aim to connect them using straight-line leaders with few crossings. These trees implicitly encode grouping constraints, as sites with a short path between their leaves must be labeled close together on the boundary. However, Klawitter et al. not only considered a different optimization function, but restricted themselves to binary trees, which cannot represent all grouping constraints. Overall, we can identify a lack of a systematic investigation of (general) grouping and ordering constraints from an algorithmic perspective in the literature. With this paper, we aim to fill this gap.

Contributions. In Section 2, we take a closer look at 1-CBL. We prove that it is NP-hard to find an admissible labeling with sliding candidates and unrestricted label heights (Section 2.1) and present polynomial-time algorithms for fixed candidates and unrestricted label heights (Section 2.2) and for sliding candidates and uniform-height labels (Section 2.3). To that end, we combine a dynamic program with a novel data structure based on PQ-Trees. As our final contribution, we show in Section 3 that 2-CBL is NP-complete, even for uniform-height labels and fixed candidates. To the best of our knowledge, this is the first two-sided boundary labeling problem that is already NP-hard in such a restricted setting. We summarize our results in Table 1.

Table 1 Our results on b-CBL with n sites, m candidates, and a family Γ of k grouping constraints.

b	candidates	label height	result	reference
1	sliding	non-uniform	NP-hard	Theorem 2.1
1	fixed	non-uniform	$\mathcal{O}(n^5m^3 \log m + k + \sum_{\mathcal{G}\in\Gamma} \lvert\mathcal{G}\rvert)$	Theorem 2.6
1	sliding	uniform	$\mathcal{O}(n^{11} \log n + k + \sum_{\mathcal{G}_p\in\Gamma} \lvert\mathcal{G}\rvert)$	Theorem 2.9
2	fixed	uniform	NP-complete	Theorem 3.1

Full proofs of statements marked by ★ are deferred to the full version [10].

2 The Constrained One-Sided Boundary Labeling Problem

In Section 2.1, we show that finding an admissible labeling for an instance of 1-CBL is NP-hard. However, by restricting the input to either fixed candidates (Section 2.2) or uniform labels (Section 2.3), we can obtain polynomial-time algorithms.

2.1 Constrained One-Sided Boundary Labeling is NP-hard

To show that it is NP-hard to find an admissible labeling in an instance of 1-CBL, we can reduce from the (weakly) NP-complete problem Partition. Inspired by a reduction by Fink and Suri [12] from Partition to the problem of finding a planar labeling with non-uniform height labels and sliding candidates in the presence of a single obstacle on the plane, we will create a gadget of sites with grouping and ordering constraints that simulates such an obstacle. The following construction is not in general position and contains leader-site crossings. We resolve this issue in the full version [10].

Construction of the Instance. We will first give a reduction that only uses grouping constraints. In the end, we show how we can replace the grouping constraints by ordering constraints. Let $(\mathcal{A} = \{a_1, \dots a_N\}, w\colon \mathcal{A} \to \mathbb{N}^+)$ be an instance of the weakly NP-complete problem Partition, in which we want to know if there exists a subset $\mathcal{A}' \subseteq \mathcal{A}$ such that $\sum_{a\in\mathcal{A}'} w(a) = \sum_{a\in\mathcal{A}\setminus\mathcal{A}'} w(a) = A$, for some integer A, for which we can safely assume $A \geq 6$. We create for each element a_i a site s_i whose corresponding label ℓ_{s_i} has a height of $w(a_i)$, and place the sites on a horizontal line next to each other. Furthermore, we create five sites, b_1 to b_5, with corresponding labels of height 1 for b_1 and b_5, height $\lfloor (A-4)/2 \rfloor$ for b_2 and b_4, and height 2 or 3 for b_3, depending on whether A is even or odd, respectively, and place them as in Figure 3. Observe that the heights of the labels for b_1 to b_5 sum up to A. We create the grouping constraints $\{\{b_1, b_2, b_3\}, \{b_2, b_3, b_4\}, \{b_3, b_4, b_5\}\}$, which enforce that any admissible labeling must label these sites as indicated in Figure 3. Since there is neither an alternative order of the labels nor room to slide around, the labels of b_2 to b_4 must be placed contiguously, without any free space, and at that fixed position on the boundary. Hence, we call the resulting structure a *block*. We create two additional blocks above and below the sites for $\mathcal{A}$ to create two A-high free windows on the boundary; see Figure 3. These windows will be the only place the labels for the sites representing the elements of $\mathcal{A}$ can be. Thus, we can form an equivalence between partitioning the elements of $\mathcal{A}$ into two sets, $\mathcal{A}_1$ and $\mathcal{A}_2$, and placing the labels for the sites in the upper or lower window on the boundary, respectively. Since the windows on the boundary have a height of A, we ensure that the sum of the label heights in each window is exactly A. Finally, note that the grouping

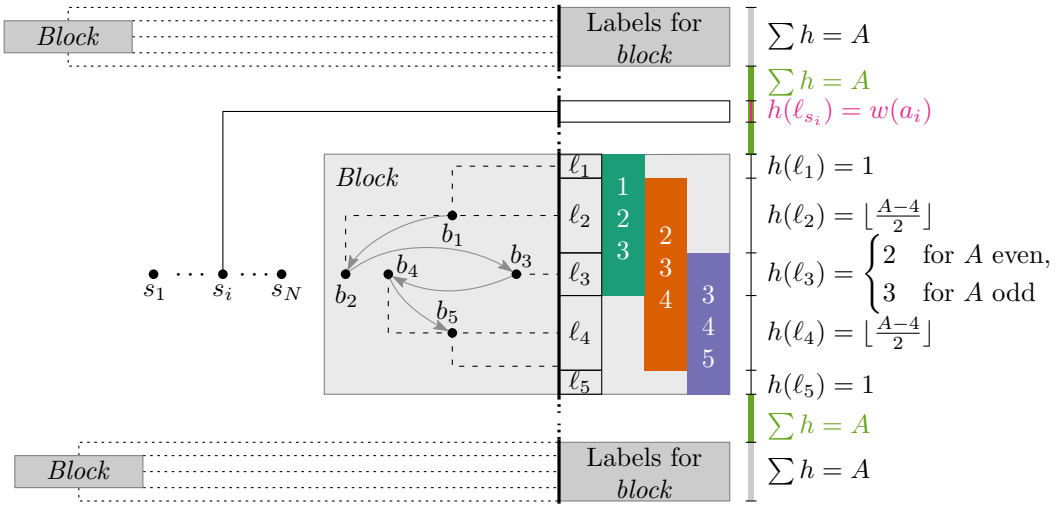


Figure 3 Components of the instance created by our NP-hardness reduction. Colored bands visualize the grouping constraints of a block, that can be replaced by ordering constraints (arrows). Note that the label heights and distances in this figure are not to scale.

constraints we used to keep the labels for the sites of the blocks in place can be exchanged by the ordering constraints $\{b_1 \preccurlyeq b_2, b_2 \preccurlyeq b_3, b_3 \preccurlyeq b_4, b_4 \preccurlyeq b_5\}$ (also shown in Figure 3 via the arrows). Similar substitutions in the other two blocks yield Theorem 2.1.

▶ **Theorem 2.1 (★).** *Deciding if an instance of* 1-CBL *has an admissible labeling is NP-hard, even for a constant number of grouping or ordering constraints.*

We conclude this section by noting that the problem Partition is only *weakly* NP-complete. Therefore, Theorem 2.1 does not prove that 1-CBL is also NP-hard in the strong sense and such a result would require a reduction from a different problem. Alternatively obtaining a pseudo-polynomial algorithm for our problem at hand would show that it is weakly NP-complete, ruling out strong NP-hardness under common complexity assumptions. We refer to the book by Garey and Johnson [15] for an introduction to NP-completeness and its flavors and leave both directions open for future work.

2.2 Fixed Candidate Reference Points

We assume that we are given a set $\mathcal{C}$ of $m \geq n$ candidates. Benkert et al. [6] observed that in a planar labeling $\mathcal{L}$, the leader λ_L connecting the leftmost site $s_L \in \mathcal{S}$ with some candidate c_L splits the instance $\mathcal{I}$ into two independent sub-instances, $\mathcal{I}_1$ and $\mathcal{I}_2$, excluding s_L and c_L. Therefore, we can describe a sub-instance I of $\mathcal{I}$ by two leaders (s_1, c_1) and (s_2, c_2) that bound the sub-instance from above and below, respectively. We denote the sub-instance as $I = (s_1, c_1, s_2, c_2)$ and refer with $\mathcal{S}(I)$ and $\mathcal{C}(I)$ to the sites and candidates in I, *excluding* those used in the definition of I, i.e., $\mathcal{S}(I) := \{s \in \mathcal{S} \mid x(s_1) < x(s),\ x(s_2) < x(s),\ y(c_2) < y(s) < y(c_1)\}$ and $\mathcal{C}(I) := \{c \in \mathcal{C} \mid y(c_2) < y(c) < y(c_1)\}$. Similarly, for a leader $\lambda = (s, c)$, we say that a site s' with $x(s) < x(s')$ is *above* λ if $y(s') > y(c)$ holds and *below* λ if $y(s') < y(c)$ holds. See also Figure 4 for an illustration of these definitions and notions.

Two more observations about admissible labelings can be made: First, λ_L never splits sites $s, s' \in \mathcal{G}$ with $s_L \notin \mathcal{G}$ for a $\mathcal{G} \in \Gamma$. Second, λ_L never splits sites $s, s' \in \mathcal{S}$ with s above λ_L and s' below λ_L, for which we have $s' \preccurlyeq s_L$, $s' \preccurlyeq s$, or $s_L \preccurlyeq s$. Now, we could immediately

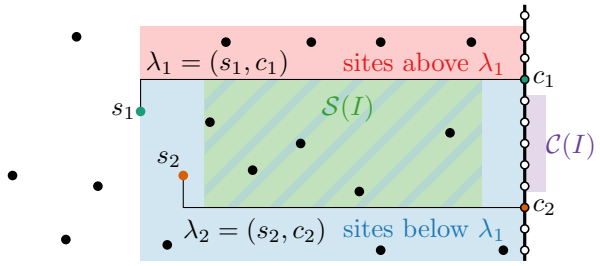


Figure 4 A sub-instance $I = (s_1, c_1, s_2, c_2)$ of our DP-algorithm and the used notation.

define a dynamic programming (DP) algorithm that evaluates the induced sub-instances for each leader that adheres to these observations. However, we would then check every constraint in each sub-instance and not make use of implicit constraints given by, for example, overlapping groups. The following data structure makes these implicit constraints explicit.

PQ-A-Graphs. Every labeling $\mathcal{L}$ induces a permutation π of the sites by reading the labels from top to bottom. Assume that we have at least one grouping constraint, i.e., $k > 0$, and let $M(\mathcal{S}, \Gamma)$ be an $n \times k$ binary matrix with $m_{i,j} = 1$ if and only if $s_i \in \mathcal{G}_j$ for $\mathcal{G}_j \in \Gamma$. We call $M(\mathcal{S}, \Gamma)$ the *sites vs. groups* matrix, and observe that $\mathcal{L}$ satisfies the constraint $\mathcal{G}_j$ if and only if the ones in the column j of $M(\mathcal{S}, \Gamma)$ are consecutive after we order the rows of $M(\mathcal{S}, \Gamma)$ according to π. If a permutation π exists such that this holds for all columns of $M(\mathcal{S}, \Gamma)$, then the matrix has the so-called *consecutive ones property* (*C1P*) [13]. This brings us to the following observation.

▶ **Observation 2.2.** *Γ are consistent for $\mathcal{S}$ if and only if $M(\mathcal{S}, \Gamma)$ has the C1P.*

Booth and Lueker [8] proposed an algorithm to check whether a binary matrix has the C1P. They use a PQ-Tree to keep track of the allowed row permutations. A *PQ-Tree* τ, for a given set $\mathcal{A}$ of elements, is a rooted tree with one leaf for each element of $\mathcal{A}$ and two different types of internal nodes t: P-nodes, that allow to freely permute the children of t, and Q-nodes, where the children of t can only be inversed [8]. Observation 2.2 tells us that each family of consistent grouping constraints can be represented by a PQ-Tree. Note that we can interpret each subtree of the PQ-Tree as a grouping constraint and we call the resulting grouping constraints *canonical groups*. However, while every canonical group is a grouping constraint, not every given grouping constraint manifests in a canonical group.

While Observation 2.2 implies that PQ-Trees can represent families of consistent grouping constraints, it is folklore that directed acyclic graphs can be used to represent partial orders, i.e., our ordering constraints. We now combine these two data structures into *PQ-A-Graphs*.

▶ **Definition 2.3** (PQ-A-Graph). *Let $\mathcal{S}$ be a set of sites, Γ be a family of consistent grouping constraints, and $\preccurlyeq$ be a partial order on $\mathcal{S}$. The* PQ-A-Graph $\mathcal{T} = (\tau, A)$ *consists of the PQ-Tree τ for Γ, on whose leaves we embed the arcs A of a directed graph representing $\preccurlyeq$.*

We denote with T_i the *subtree* in the underlying PQ-Tree τ rooted at the node t_i and with leaves(T_i) the leaf set of T_i. Figure 5 shows a PQ-A-Graph and the introduced terminology. Furthermore, observe that checking on the consistency of $(\Gamma, \preccurlyeq)$ is equivalent to solving the Reorder problem on τ and $\preccurlyeq$, i.e., asking whether we can re-order leaves(τ) such that the order induced by reading them from left to right extends the partial order $\preccurlyeq$ [22].

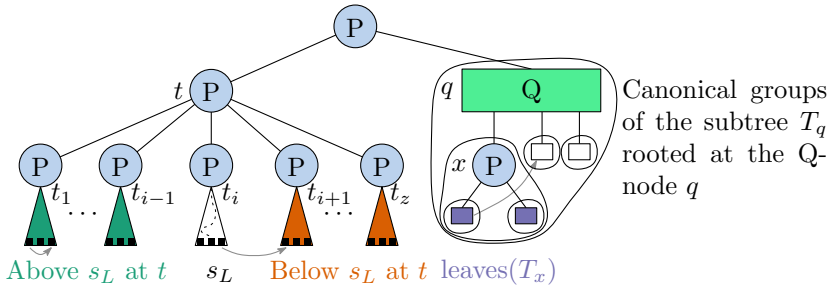


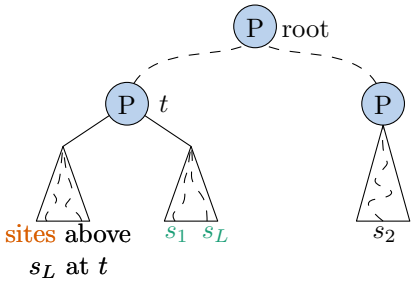
Figure 5 A sample PQ-A-Graph together with the used terminology. Leaves are indicated by squares and ordering constraints by the arrows.

▶ **Lemma 2.4 (★).** *Let $\mathcal{S}$ be a set of n sites, Γ be k grouping constraints and $\preccurlyeq$ be r ordering constraints. We can check whether $(\Gamma, \preccurlyeq)$ are consistent for $\mathcal{S}$ and, if so, create the PQ-A-Graph $\mathcal{T}$ in $\mathcal{O}(n + k + r + \sum_{\mathcal{G}\in\Gamma} |\mathcal{G}|)$ time. $\mathcal{T}$ uses $\mathcal{O}(n + r)$ space.*

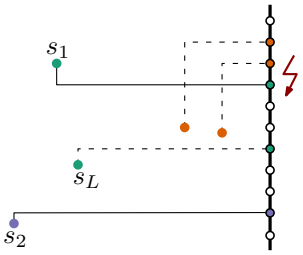
The Dynamic Programming Algorithm. Let $I = (s_1, c_1, s_2, c_2)$ be a sub-instance and s_L the leftmost site in $\mathcal{S}(I)$. Furthermore, let $\mathcal{T}(s_1, s_2)$ denote the subgraph of the PQ-A-Graph $\mathcal{T}$ rooted at the lowest common ancestor of s_1 and s_2 in $\mathcal{T}$. Note that $\mathcal{T}(s_1, s_2)$ contains at least the sites in $\mathcal{S}(I)$, together with s_1 and s_2. Therefore, it contains all constraints relevant for the sub-instance I. Other constraints either do not affect sites in I, i.e., are represented by other parts of $\mathcal{T}$, or are trivially satisfied. In particular, observe that all constraints induced by nodes further up in $\mathcal{T}$ affect a super-set of $\mathcal{S}(I)$ and in particular have been checked in some sub-instance I' that contains I, i.e., that contains s_1, s_2, c_1, and c_2. Now imagine that we want to place the label ℓ_L for s_L at the candidate $c_L \in \mathcal{C}(I)$. We have to ensure that $\lambda_L = (s_L, c_L)$ does not violate planarity with respect to the already fixed labeling and that in the resulting sub-instances there are enough candidates for the respective sites. Let Admissible$(I, \mathcal{T}, c_L)$ be a procedure that checks this and, in addition, verifies that c_L respects the constraints expressed by $\mathcal{T}(s_1, s_2)$. To do the latter efficiently, we make use of the procedure RespectsConstraints$(I, \mathcal{T}, \lambda_L)$ defined as follows.

Let t_L be the leaf for s_L in $\mathcal{T}(s_1, s_2)$. There is a unique path from t_L to root$(\mathcal{T}(s_1, s_2))$, which we traverse bottom up and consider each internal node on it. Let t be such a node with the children $t_1, \ldots, t_z$ in this order from left to right. Let T_i, $1 \leq i \leq z$, be the subtree that contains the site s_L, rooted at t_i. The labels for all sites represented by leaves$(T_1), \ldots,$ leaves(T_{i-1}) will be placed above ℓ_L in any labeling $\mathcal{L}$ of $\mathcal{S}$ in which the children of t are ordered as stated. Therefore, we call these sites *above* s_L (at t). Analogously, the sites represented by leaves$(T_{i+1}), \ldots,$ leaves(T_z) are *below* s_L (at t). Figure 5 illustrates this. The sites represented by leaves(T_i) are neither above nor below s_L at t. It is important to note that we use two different notions of *above/below*. On the one hand, sites can be above a node t in the PQ-A-Graph $\mathcal{T}(s_1, s_2)$, which depends on the order of the children of t. On the other hand, a site can also be above a leader λ, which depends on the (geometric) position of λ and is independent of $\mathcal{T}(s_1, s_2)$. Recall Figure 5 and compare it with Figure 4 for the former and latter notion of above and below, respectively.

If t is a P-node, we seek a permutation π of the children $t_1, \ldots, t_z$ of t in which *all* the sites in $\mathcal{S}(I)$ above s_L at t (in the permutation π) are above λ_L, and all the sites in $\mathcal{S}(I)$ below s_L at t (in the permutation π) are below λ_L. This means that it cannot be the case that some sites of the same subtree T that does not contain s_L are above λ_L, while others are below λ_L, as this implies that we violate the canonical grouping constraint induced by T.



(a) The PQ-A-Graph $\mathcal{T}(s_1, s_2)$, rooted at root$(\mathcal{T}(s_1, s_2))$, with $s_1 \in$ leaves(T_i).



(b) A (sub-)instance I, where a wrong permutation of the children of t in $\mathcal{T}(s_1, s_2)$ from **(a)** would label the orange sites outside I.

Figure 6 In this situation, C_{above} must not contain subtrees with sites from the sub-instance.

To not iterate through all possible permutations, we distribute the children of t, except t_i, into two sets, C_{above} and C_{below}, depending on whether the sites they represent should be above or below s_L at t. Unless specified otherwise, whenever we mention in the following the site s_1, then the same applies to s_2, but possibly after exchanging *above* with *below*.

If T_j, rooted at a child t_j of t, $1 \leq j \leq z$, $i \neq j$, only contains sites from $\mathcal{S}(I)$, we check whether all the sites are above λ_L, or if all are below λ_L. In the former case, we put t_j in the set C_{above}, and in the latter case in C_{below}. If neither of these cases applies, we return with failure as λ_L splits a (canonical) group to which s_L does not belong. If T_j contains only sites outside the sub-instance and not s_1, we immediately put it in C_{above}. However, if T_j contains $s_1 \in$ leaves(T_j), it can contain some sites in I and others outside I. As s_1 is above s_L at t by the definition of I, we must put t_j in C_{above}. Hence, we check whether all sites in leaves$(T_j) \cap \mathcal{S}(I)$ are above λ_L, i.e., whether they are indeed above s_L at t. Furthermore, if $s_1 \in$ leaves(T_i) holds, then C_{above} must not contain a child t_j containing sites from $\mathcal{S}(I)$, as they would then be labeled outside I, violating the definition of I; see Figure 6. Once t is the root of $\mathcal{T}(s_1, s_2)$, sites from T_j but outside I can be above or below s_L at t, as s_1 *and* s_2 are in the subtree rooted at t. If T_j does not contain the sites s_1 and s_2, and only sites outside I, then the leaders from s_1 or s_2 separate the sites from T_j and $\mathcal{S}(I)$. As these sites together were part of a bigger instance I' (that contains I), for which we already ensured that potential constraints relating a site from T_j and one from $\mathcal{S}(I)$ are respected, we can ignore t_j. In any other case, we perform as described above.

The checks that have to be performed if t is a Q-node are conceptually the same, but simpler, since Q-nodes only allow to inverse the order: Either all sites above s_L at t are above λ_L, and all sites below s_L at t are below λ_L, or all sites above s_L at t are below λ_L, and all sites below s_L at t are above λ_L. In the former case, we keep the order of the children at the node t as they are. In the latter case, we inverse the order of the children at the node t. Note that if a child of t contains the sites s_1, s_2, or sites outside the sub-instance I but in $\mathcal{T}(s_1, s_2)$, one of the two allowed inversions is enforced by the definition of I.

Until now we only verified that we adhere to the grouping constraints. To ensure that we do not violate an ordering constraint, we maintain a look-up table that stores for each site whether it belongs to C_{above}, C_{below}, or T_i. Then, we check for each of the ordering constraints in $\mathcal{T}(s_1, s_2)$ in constant time whether we violate it or not. Note that we violate an ordering constraint if the corresponding arc runs from C_{below} to T_i or to C_{above}, or from T_i to C_{above}. We observe that we query the position of each site s $\mathcal{O}(1)$ times and determine each time its position with respect to the leader λ_L or check whether it is in the sub-instance.

Afterwards, we do not consider this site anymore. As each ordering constraint can be checked in $\mathcal{O}(1)$ time, we have an overall running time of $\mathcal{O}(n + r)$ for the checks at a node t of $\mathcal{T}(s_1, s_2)$, which already includes the computation of the look-up tables.

We say that c_L *respects the constraints for* s_L imposed by $\mathcal{T}(s_1, s_2)$ in the sub-instance $I = (s_1, c_1, s_2, c_2)$ if it respects them at every node t on the path from s_L to the root of $\mathcal{T}(s_1, s_2)$. RespectsConstraints$(I, \mathcal{T}, \lambda_L)$ performs these checks for each node on the path from s_L to root$(\mathcal{T}(s_1, s_2))$. The length of this path is bounded by the depth of $\mathcal{T}(s_1, s_2)$ which is in $\mathcal{O}(n)$. Hence, RespectsConstraints$(I, \mathcal{T}, \lambda_L)$ runs in $\mathcal{O}(n\,(n + r))$ time. In the following lemma, we show that Admissible$(I, \mathcal{T}, c_L)$ takes $\mathcal{O}(n^2 + nr + \log m)$ time.

▶ **Lemma 2.5 (★).** *Let $I = (s_1, c_1, s_2, c_2)$ be a sub-instance of our DP-Algorithm with the constraints expressed by a PQ-A-Graph $\mathcal{T}$. We can check whether the candidate $c_L \in \mathcal{C}(I)$ is admissible for the leftmost site $s_L \in \mathcal{S}(I)$ using* Admissible$(I, \mathcal{T}, c_L)$ *in $\mathcal{O}(n^2 + nr + \log m)$ time, where $n = |\mathcal{S}|$, $m = |\mathcal{C}|$, and r is the number of ordering constraints.*

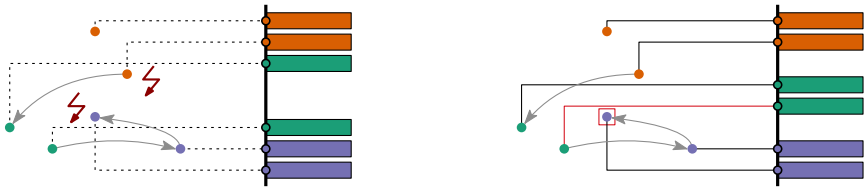
For a sub-instance $I = (s_1, c_1, s_2, c_2)$, we store in a table D the value $f(\mathcal{L}^*)$ of an optimal admissible labeling $\mathcal{L}^*$ on I or ∞ if none exists. If I does not contain a site we set $D[I] = 0$. Otherwise, we use the following relation, where the minimum of the empty set is ∞.

$$D[I] = \min_{\substack{c_L \in \mathcal{C}(I) \text{ where} \\ \textsc{Admissible}(I, \mathcal{T}, c_L) \text{ is true}}} (D[(s_1, c_1, s_L, c_L)] + D[(s_L, c_L, s_2, c_2)]) + f((s_L, c_L))$$

To show correctness of our DP-Algorithm, one can use a proof analogous to the one of Benkert et al. [6], who propose a similar dynamic program to compute a one-sided labeling with *po*-leaders and a similar-structured optimization function, combined with the fact that we consider only those candidates that are admissible for s_L. By adding artificial sites s_0 and s_{n+1}, and candidates c_0 and c_{m+1}, that bound the instance from above and below, we can describe any sub-instance by a tuple $I = (s_1, c_1, s_2, c_2)$, and in particular the sub-instance for $\mathcal{I}$ by $I_0 = (s_0, c_0, s_{n+1}, c_{m+1})$. As two sites and two candidates describe a sub-instance, there are up to $\mathcal{O}(n^2m^2)$ possible sub-instances to evaluate. We then fill the table D top-down using memoization. This guarantees us that we have to evaluate each sub-instance I at most once, and only those that arise from admissible candidates. The running time of evaluating a single sub-instance is dominated by the time required to determine for each candidate whether it is admissible. Combined with the size of the table D, we get the following.

▶ **Theorem 2.6 (★).** 1-CBL *for n sites, m* fixed candidates*, r ordering, and k grouping constraints Γ can be solved in $\mathcal{O}(n^5m^3 \log m + k + \sum_{\mathcal{G} \in \Gamma} |\mathcal{G}|)$ time and $\mathcal{O}(n^2m^2)$ space.*

Real-world instances often consist of less than 50 sites [26] and we do not expect the number of candidates to be significantly larger than the number of sites. Hence, the running time of Theorem 2.6 does not immediately rule out the practical applicability of our results. Indeed, initial experiments for uniform-height labels confirmed that our algorithm terminates within a few seconds for instances with realistic sizes [1, 16] of up to 25 sites and 50 candidates; see the full version [10] for details and Figure 1b for an example. In fact, dynamic programming is frequently used to obtain exact polynomial-time algorithms in external labeling [5] and it is not uncommon that such algorithms have high running times of up to $\mathcal{O}(n^6)$ and $\mathcal{O}(n^9)$ for the one- and two-sided setting with *po*-leaders, respectively [6, 12, 21, 23]. Finally, we observed in our experiments that the position of the candidates influenced the feasibility of an instance, which makes considering sliding candidates interesting and relevant.



(a) We cannot avoid crossings if we want to respect the constraints.

(b) An alternative set of candidates to **(a)** which allows for an admissible labeling.

Figure 7 An instance whose admissibility depends on the position of the candidates.

2.3 Sliding Candidate Reference Points with Uniform-Height Labels

Fixed candidates have the limitation that the admissibility of an instance depends on the choice and position of the candidates, as Figure 7 shows. By allowing the labels to slide along a sufficiently long vertical boundary line, we remove this limitation. To avoid the NP-hardness shown in Section 2.1, we require that all labels now have uniform height $h > 0$.

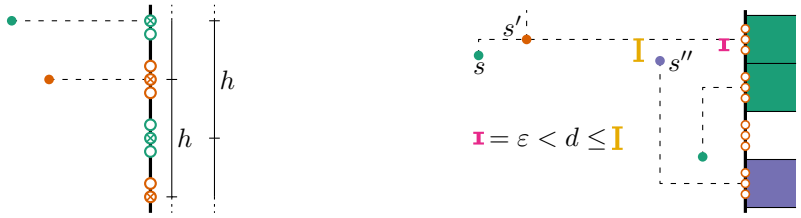
In this section, we will define for each site s a set of $\mathcal{O}(n)$ candidates placed at multiples of h away from $y(s)$, building on an idea of Fink and Suri [12] shown in Figure 8a with the crossed candidates. After extending them by some offset $\varepsilon > 0$ we will prove in order: That if an instance has an admissible labeling, it also has an admissible (bend-minimal) labeling using these candidates (Lemma 2.7), that if such an instance has an admissible labeling in which every leader has a minimum distance to every non-incident site, then there is a labeling with the same property using a slightly different set of candidates, which also has equal or smaller total leader length (Lemma 2.8) and that these results in concert with Theorem 2.6 can be used to solve 1-CBL with uniform-height labels in polynomial time (Theorem 2.9).

Let d be defined as $d := \min_{s,s' \in \mathcal{S}} (|y(s) - y(s')| - qh)$, where $q = \lfloor |y(s) - y(s')| / h \rfloor$, i.e., the smallest distance one must move some site (down) in the instance such that it is vertically a multiple of h away from some other site. For the following arguments to work, we require $d > 0$. However, this can easily be ensured by enforcing that the vertical distance $|y(s) - y(s')|$ between any pair of sites s and s' is not a multiple of h, which can be achieved by perturbating some sites slightly. As we then have $d > 0$, we can select an ε with $0 < \varepsilon < d$. We now define a set of $\mathcal{O}(n^2)$ candidates such that there exists an admissible labeling on these (fixed) candidates, if the instance with sliding candidates possesses an admissible labeling. For each $s \in \mathcal{S}$, we define the set $\mathcal{C}(s) := \{y(s)+ih,\ y(s)+ih\pm\varepsilon,\ y(s)-ih,\ y(s)-ih\pm\varepsilon \mid 0 \leq i \leq n\}$ of candidates. Now, we define the set of *canonical candidates* $\mathcal{C}(\mathcal{S})$ as $\mathcal{C}(\mathcal{S}) := \bigcup_{s \in \mathcal{S}} \mathcal{C}(s)$, some of which are depicted in Figure 8a, and show the following.

▶ **Lemma 2.7 (★).** *Let $\mathcal{I}$ be an instance of* 1-CBL *with uniform-height labels and sliding candidates. If $\mathcal{I}$ possesses an admissible labeling, it also has one with candidates from $\mathcal{C}(\mathcal{S})$.*

Proof Sketch. Our proof builds on arguments used by Fink and Suri for a similar result [12] and we call a maximal set of touching (but non-overlapping) labels a *stack* [27]. The idea is to transform an admissible labeling $\mathcal{L}$ into an admissible labeling $\mathcal{L}'$ in which each candidate is from $\mathcal{C}(\mathcal{S})$. Labels at a candidate above the candidate $c \in \mathcal{C}(\mathcal{S})$ with $y(c) = y(s_t) + h$ for the top-most site s_t are arranged in $\mathcal{L}'$ as a single stack so that the bottom-most candidate coincides with c. We arrange labels at a candidate at least h below the bottom-most site symmetrically. As they are located above and below all sites, this cannot introduce crossings.

For the remaining labels in $\mathcal{L}$, we iteratively take the bottom-most not yet moved label ℓ and move it upwards until we either hit a candidate from $\mathcal{C}(\mathcal{S})$ or another label ℓ'. In the latter case, we "merge" ℓ and ℓ' into a stack and move them from now on simultaneously. We



(a) Induced candidates as in [12] (marked by a cross) and the extension to canonical candidates.

(b) The leader of s crosses s' after moving labels upwards. We show the candidates from $\mathcal{C}(s')$.

Figure 8 The set of reference points we construct and **(b)** their usage in the proofs.

continue moving the remaining labels in the same manner. Observe that we never move a label past a site, but might introduce leader-site crossings. They can arise if in $\mathcal{L}$ a leader λ from a site s passes between a candidate of $\mathcal{C}(\mathcal{S})$ and a site $s' \neq s$. There, the first candidate that we hit for ℓ might be induced by s' and λ could now cross s'. In such a case, we take that label and its potential stack and move it downwards until we hit a candidate from $\mathcal{C}(\mathcal{S})$. By our selection of ε, it is guaranteed that we will hit a candidate before any other site, since there is at least one other candidate strictly between the end of the stack and any other site s''. We already indicated this in Figure 8a, but show it in more detail in Figure 8b with the orange candidates. After moving all such labels simultaneously, we obtain the labeling $\mathcal{L}'$ where each label is at a candidate from $\mathcal{C}(\mathcal{S})$. Since the relative placement of the labels in $\mathcal{L}'$ is identical to $\mathcal{L}$, all constraints are still respected and $\mathcal{L}'$ is admissible. ◀

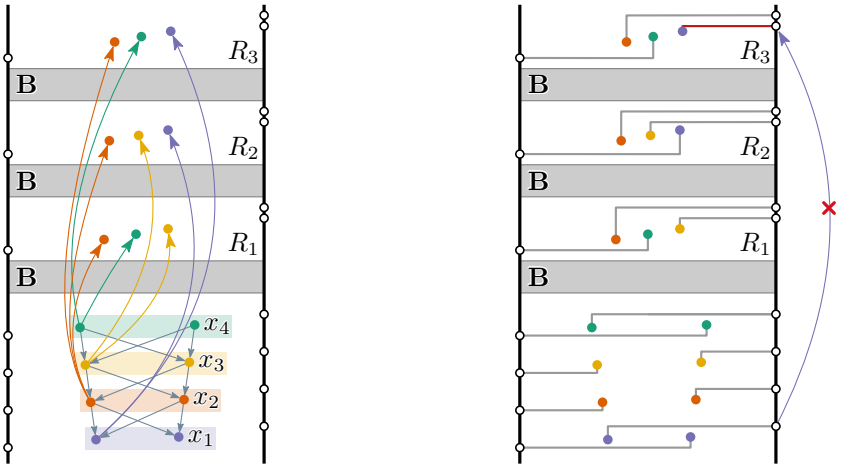
Observe that if we want to obtain a labeling with the minimum number of bends, then it only matters whether a site s is labeled at a candidate c with $y(s) = y(c)$. Labeling s at any other candidate $c' \neq c$ contributes with one bend to the labeling. As each site $s \in \mathcal{S}$ induces a candidate $c \in \mathcal{C}(\mathcal{S})$ with $y(s) = y(c)$, our set of canonical candidates allows for a bend-minimal labeling. For length-minimal labelings, instances without optimal labelings exist. See for example Figure 7b, where we can always move the red leader by a small $\varepsilon' > 0$ closer to the purple site marked with a red box. To ensure the existence of length-minimal labelings, we enforce that the leaders maintain a minimum vertical distance of $v_{\min} > 0$ to non-incident sites. We define $\mathcal{C}'(s) := \{y(s)+qh,\ y(s)+qh \pm v_{\min},\ y(s)-qh,\ y(s)-qh \pm v_{\min} \mid 0 \leq q \leq n\}$, which is an alternative set of canonical candidates that takes $v_{\min}$ into account. Equipped with $\mathcal{C}'(\mathcal{S}) := \bigcup_{s \in \mathcal{S}} \mathcal{C}'(s)$, we can show Lemma 2.8, which is a variant of Lemma 2.7.

▶ **Lemma 2.8 (★).** *Let $\mathcal{I}$ be an instance of* 1-CBL *with uniform-height labels and where the leaders must maintain a vertical distance of at least $v_{\min}$ to non-incident sites. If $\mathcal{I}$ possesses an admissible labeling $\mathcal{L}$ with sliding candidates, then we can find an admissible labeling $\mathcal{L}'$ with candidates from $\mathcal{C}'(\mathcal{S})$ such that the leader length of $\mathcal{L}'$ is at most the one of $\mathcal{L}$.*

Note that $\mathcal{C}'(\mathcal{S})$ can contain candidates for which leaders would not satisfy the requirement on a vertical distance of at least $v_{\min}$ to non-incident sites. Recall that we never moved past a candidate while sliding labels and created in $\mathcal{C}'(\mathcal{S})$ candidates that are $v_{\min}$ away from sites. Hence, we ensure that $\mathcal{L}'$ satisfies this additional criterion as well. This criterion can also be patched into the DP-algorithm without affecting its running time. Finally, this additional criterion is not a limitation, as this is already often required in real-world labelings [26].

With Lemmas 2.7 and 2.8 at hand, we can show the following theorem.

▶ **Theorem 2.9.** 1-CBL *for n sites with* uniform-height labels*, k grouping, and r ordering constraints can be solved in $\mathcal{O}(n^{11} \log n + k + \sum_{\mathcal{G} \in \Gamma} |\mathcal{G}|)$ time and $\mathcal{O}(n^6)$ space.*



(a) The constructed instance.

(b) A (non-admissible) labeling that violates an ordering constraint.

Figure 9 The instance created by our reduction for the formula $R_1 \wedge R_2 \wedge R_3 = (x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4)$. The variable gadgets for x_1, x_2, x_3, and x_4 as well as their occurrences in clause gadgets are drawn in blue, red, yellow, and green, respectively. Ordering constraints between variables and their occurrences in clauses are indicated in (a). If two variables of a clause are set to true, we violate at least one ordering constraint as highlighted with the red leader for the purple site in the clause R_3 in (b). The labeling from (b) induces the variable assignment $x_1 = x_4 =$ true and $x_2 = x_3 =$ false, which does not satisfy $R_3 = (x_1 \vee x_2 \vee x_4)$ as x_1 and x_4 are set to true.

Proof. We note that $\mathcal{C}(\mathcal{S})$ and $\mathcal{C}'(\mathcal{S})$ consist of $\mathcal{O}(n^2)$ canonical candidates. Lemmas 2.7 and 2.8 ensure that we can obtain on these candidates an admissible, bend-, and length-minimal labeling, if there exists one at all. Thus, we can use our DP-Algorithm from Section 2.2 and obtain Theorem 2.9 by plugging in $m = \mathcal{O}(n^2)$ in Theorem 2.6. ◀

3 Constrained Two-Sided Boundary Labeling is NP-complete

In the previous section, we showed that 1-CBL, while generally NP-hard, can be solved efficiently if we have either fixed candidates or labels of uniform height. This does not extend to the generalization of the problem to two sides, as the following theorem underlines.

▶ **Theorem 3.1** (★). *Deciding if an instance of* 2-CBL *has an admissible labeling is NP-complete, even for uniform-height labels and fixed candidates.*

Proof Sketch. NP-membership follows from the definition of admissibility, and NP-hardness can be shown by reducing from Positive 1-In-3 Sat, see Figure 9 for an example. The main building block is the *blocker gadget* B, which consists of eight sites and uses grouping and ordering constraints to let the orthogonal parts of its leaders span the entire width between the two boundaries. This enforces that sites on one side of the gadget cannot be labeled on the other side and vice versa. We use this to divide the space between the vertical boundaries into separate strips: one per clause R_i and one at the bottom for all variables. The former contains three *clause sites*, one per occurring variable, of which only one can be labeled on the left side. The latter contains per variable two *variable sites* which encode if the variable is true (a site is labeled on the right side) or false (it is labeled on the left side). Ordering constrains force any variable site to be labeled above any corresponding clause site. The

blockers make this impossible, thus forcing all clause sites to be labeled on the opposite side of their variable site, creating a correspondence to a consistent variable assignment. Finally, as in every clause strip, there is only one candidate on the left side, exactly one clause site can be labeled on the left, corresponding to the (single) variable satisfying this clause. ◀

4 Conclusion

We introduced and studied grouping and ordering constraints in boundary labeling. While finding an admissible labeling is NP-hard in general, polynomial-time algorithms for one-sided instances with fixed candidates or uniform-height labels exist. Future work could try to speed up the admissibility checks in our dynamic program to reduce its overall running time or investigate the incorporation of *soft* constraints, i.e., consider the task of maximizing the number of satisfied constraints. Since we can also label features other than points, it is worth studying a variant of this problem with uncertain or variable site locations. Similarly, the support of other leader types or entire other external labeling styles should be investigated.

References

1 Lukas Barth, Andreas Gemsa, Benjamin Niedermann, and Martin Nöllenburg. On the readability of leaders in boundary labeling. *Information Visualization*, 18(1):110–132, 2019. doi:10.1177/1473871618799500.

2 Michael A. Bekos, Sabine Cornelsen, Martin Fink, Seok-Hee Hong, Michael Kaufmann, Martin Nöllenburg, Ignaz Rutter, and Antonios Symvonis. Many-to-One Boundary Labeling with Backbones. *Journal of Graph Algorithms and Applications (JGAA)*, 19(3):779–816, 2015. doi:10.7155/jgaa.00379.

3 Michael A. Bekos, Michael Kaufmann, Martin Nöllenburg, and Antonios Symvonis. Boundary Labeling with Octilinear Leaders. *Algorithmica*, 57(3):436–461, 2010. doi:10.1007/s00453-009-9283-6.

4 Michael A. Bekos, Michael Kaufmann, Antonios Symvonis, and Alexander Wolff. Boundary labeling: Models and efficient algorithms for rectangular maps. *Computational Geometry*, 36(3):215–236, 2007. doi:10.1016/j.comgeo.2006.05.003.

5 Michael A. Bekos, Benjamin Niedermann, and Martin Nöllenburg. *External Labeling: Fundamental Concepts and Algorithmic Techniques*. Synthesis Lectures on Visualization. Springer, 2021. doi:10.1007/978-3-031-02609-6.

6 Marc Benkert, Herman J. Haverkort, Moritz Kroll, and Martin Nöllenburg. Algorithms for Multi-Criteria Boundary Labeling. *Journal of Graph Algorithms and Applications (JGAA)*, 13(3):289–317, 2009. doi:10.7155/jgaa.00189.

7 Satyam Bhuyan and Santanu Mukherjee (sciencefacts.net). Layers of the Sun, 2023. Accessed on 2023-09-07. URL: https://www.sciencefacts.net/layers-of-the-sun.html.

8 Kellogg S. Booth and George S. Lueker. Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-Tree Algorithms. *Journal of Computer and System Sciences (JCSS)*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.

9 Mary Helen Briscoe. *A Researcher's Guide to Scientific and Medical Illustrations*. Springer Science & Business Media, 1990. doi:10.1007/978-1-4684-0355-8.

10 Thomas Depian, Martin Nöllenburg, Soeren Terziadis, and Markus Wallinger. Constrained Boundary Labeling, 2024. doi:10.48550/arXiv.2402.12245.

11 Martin Fink, Jan-Henrik Haunert, André Schulz, Joachim Spoerhase, and Alexander Wolff. Algorithms for Labeling Focus Regions. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2583–2592, 2012. doi:10.1109/TVCG.2012.193.

12 Martin Fink and Subhash Suri. Boundary Labeling with Obstacles. In *Proc. 28th Canadian Conference on Computational Geometry (CCCG)*, pages 86–92. Simon Fraser University, 2016.

13 Delbert Fulkerson and Oliver Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965.

14 Stadtförsterei Fürth. Altersbestimmung und Baumanatomie, 2021. URL: `https://www.stadtwald.fuerth.de/waldlehrpfad/baumanatomie-und-altersbestimmung`. Accessed on 2024-04-16.

15 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

16 Sven Gedicke, Lukas Arzoumanidis, and Jan-Henrik Haunert. Automating the external placement of symbols for point features in situation maps for emergency response. *Cartography and Geographic Information Science (CaGIS)*, 50(4):385–402, 2023. `doi:10.1080/15230406.2023.2213446`.

17 Sven Gedicke, Annika Bonerath, Benjamin Niedermann, and Jan-Henrik Haunert. Zoomless Maps: External Labeling Methods for the Interactive Exploration of Dense Point Sets at a Fixed Map Scale. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1247–1256, 2021. `doi:10.1109/TVCG.2020.3030399`.

18 Andreas Gemsa, Jan-Henrik Haunert, and Martin Nöllenburg. Multirow Boundary-Labeling Algorithms for Panorama Images. *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, 1(1):1:1–1:30, 2015. `doi:10.1145/2794299`.

19 Timo Götzelmann, Knut Hartmann, and Thomas Strothotte. Agent-Based Annotation of Interactive 3D Visualizations. In *Proc. 6th International Symposium on Smart Graphics (SG)*, volume 4073 of *Lecture Notes in Computer Science (LNCS)*, pages 24–35. Springer, 2006. `doi:10.1007/11795018_3`.

20 Timo Götzelmann, Knut Hartmann, and Thomas Strothotte. Contextual Grouping of Labels. In *Proc. 17th Simulation und Visualisierung (SimVis)*, pages 245–258. SCS Publishing House e.V., 2006. URL: `http://www.simvis.org/Tagung2006/sv-proceedings.html`.

21 Philipp Kindermann, Benjamin Niedermann, Ignaz Rutter, Marcus Schaefer, André Schulz, and Alexander Wolff. Multi-sided Boundary Labeling. *Algorithmica*, 76(1):225–258, 2016. `doi:10.1007/s00453-015-0028-4`.

22 Pavel Klavík, Jan Kratochvíl, Yota Otachi, Toshiki Saitoh, and Tomás Vyskocil. Extending Partial Representations of Interval Graphs. *Algorithmica*, 78(3):945–967, 2017. `doi:10.1007/s00453-016-0186-z`.

23 Jonathan Klawitter, Felix Klesen, Joris Y. Scholl, Thomas C. van Dijk, and Alexander Zaft. Visualizing Geophylogenies - Internal and External Labeling with Phylogenetic Tree Constraints. In *Proc. 12th International Conference Geographic Information Science (GIScience)*, volume 277 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. `doi:10.4230/LIPIcs.GIScience.2023.5`.

24 Konrad Mühler and Bernhard Preim. Automatic Textual Annotation for Surgical Planning. In *Proc. 14th International Symposium on Vision, Modeling, and Visualization (VMV)*, pages 277–284. DNB, 2009.

25 Benjamin Niedermann and Jan-Henrik Haunert. Focus+context map labeling with optimized clutter reduction. *International Journal of Cartography*, 5(2–3):158–177, 2019. `doi:10.1080/23729333.2019.1613072`.

26 Benjamin Niedermann, Martin Nöllenburg, and Ignaz Rutter. Radial Contour Labeling with Straight Leaders. In *Proc. 10th IEEE Pacific Visualization Symposium (PacificVis)*, PacificVis '17, pages 295–304. IEEE Computer Society, 2017. `doi:10.1109/PACIFICVIS.2017.8031608`.

27 Martin Nöllenburg, Valentin Polishchuk, and Mikko Sysikaski. Dynamic one-sided boundary labeling. In *Proc. 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS)*, GIS '10, pages 310–319. Association for Computing Machinery (ACM), 2010. `doi:10.1145/1869790.1869834`.

28 Markus Tatzgern, Denis Kalkofen, and Dieter Schmalstieg. Dynamic Compact Visualizations for Augmented Reality. In *Proc. 20th IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 3–6. IEEE Computer Society, 2013. `doi:10.1109/VR.2013.6549347`.

29 Ian Vollick, Daniel Vogel, Maneesh Agrawala, and Aaron Hertzmann. Specifying label layout style by example. In *Proc. 20th ACM Symposium on User Interface Software and Technology (UIST)*, UIST '07, pages 221–230. Association for Computing Machinery (ACM), 2007. doi:10.1145/1294211.1294252.

30 Colin Ware. *Chapter 6 – Static and Moving Patterns*, pages 179–237. Elsevier, 2013. doi:10.1016/b978-0-12-381464-7.00006-5.