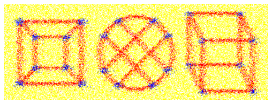

Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 13, no. 3, pp. 289–317 (2009)

Algorithms for Multi-Criteria Boundary Labeling

Marc Benkert[1] *Herman Haverkort*[2] *Moritz Kroll*[1]
Martin Nöllenburg[1]

[1]Faculty of Informatics,
Karlsruhe University and Karlsruhe Institute of Technology (KIT),
Germany
[2]Department of Mathematics and Computer Science,
Eindhoven University of Technology,
The Netherlands

Abstract

We present new algorithms for labeling a set P of n points in the plane with labels that are aligned to one side of the bounding box of P. The points are connected to their labels by curves (leaders) that consist of two segments: a horizontal segment, and a second segment at a fixed angle with the first. Our algorithms find a collection of crossing-free leaders that minimizes the total number of bends, the total length, or any other 'badness' function of the leaders. A generalization to labels on two opposite sides of the bounding box of P is considered and an experimental evaluation of the performance is included.

Submitted:	Reviewed:	Revised:	Accepted:
December 2007	June 2008	August 2008	November 2008
	Final:	Published:	
	December 2008	November 2009	
Article type:		Communicated by:	
Regular paper		S.-H. Hong and T. Nishizeki	

This work was started on the 9th "Korean" Workshop on Computational Geometry and Geometric Networks at Schloss Dagstuhl, Germany, 2006. Research supported in part by the German Science Foundation (DFG) under grant WO 758/4-2 and 4-3 and by the EU under grant DELIS (contract no. 001907).

E-mail addresses: mbenkert@iti.uka.de (Marc Benkert) cs.herman@haverkort.net (Herman Haverkort) moritz.kroll@gmx.de (Moritz Kroll) noellenburg@iti.uka.de (Martin Nöllenburg)

1 Introduction

Presentations of visual information often make use of textual labels for features of interest within the visualizations. Examples are found in diverse areas such as cartography, anatomy, engineering, sociology, etc. Graphics in these areas may have very dense regions in which objects need textual labels to be fully understood. A lot of research on automatic label placement has concentrated on placing labels inside the graphic itself, see the bibliography on map labeling by Wolff and Strijk [8]. However, this is not always possible: sometimes the labels are too large, the labeled features lie too close to each other, or the underlying graphic should remain fully visible. In such cases it is often necessary to place the labels next to the actual illustration and connect each label to its associated object by a curve as in Figure 1. This is also denoted as a *call-out*, and the curves are called *leaders*. Geographic maps that depict metropolitan areas and medical atlases often use call-outs, see Figure 2 for an example of the latter.

To produce a call-out, we have to decide where exactly to place each object's label and how to draw the curves such that the connections between objects and labels are clear and the leaders do not clutter the figure. Clearly, leaders should not intersect each other to avoid confusion, and several authors have designed algorithms to produce non-intersecting leaders in several settings. Fekete and Plaisant [6] label point objects with polygonal leaders with up to two bends in an interactive setting. Ali et al. [1] describe heuristics to label points with straight-line and rectilinear leaders. Bekos et al. use rectilinear leaders with up to two bends. They study settings with labels arranged on one, two, or four sides of the bounding box of the illustration [5], in multiple stacks to the left [3], or where the objects to be labeled are polygons rather than points [4]. The optimization criteria that they consider are minimizing the total length and minimizing the total number of bends.

Maybe surprisingly, relying exclusively on straight-line leaders is not always the best choice. The reason is that the variety of different slopes among the leaders may clutter the figure, especially if the number of labels is large. An example of this effect is shown in Figure 2. Leaders tend to look less disturbing if their shape is more uniform and a small number of slopes is used, like with rectilinear leaders. On the other hand, leaders appear easier to follow if their bends are smooth, as illustrated in Figure 3; thus 90° angles may rather be avoided. In this work we focus on how to label points with labels on one side of the illustration and leaders with at most one bend. Bekos et al. [5] studied

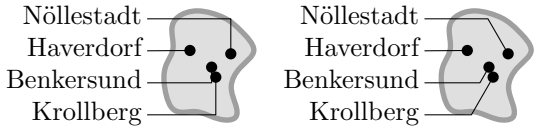


Figure 1: Examples of call-outs with bends of 90° (*po*-leaders) or 120° (*do*-leaders), respectively. The leaders for Haverdorf are *direct* leaders.

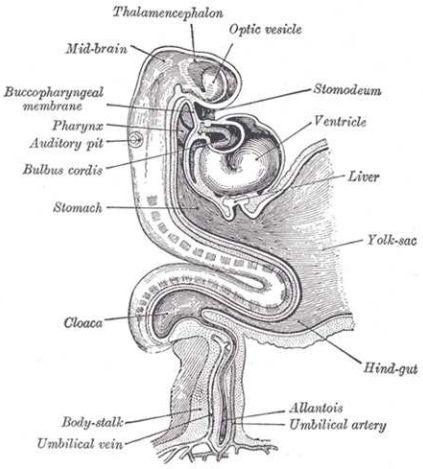


Figure 2: Anatomy of a human embryo labeled with direct leaders [7].

Figure 3: Weather forecast map by DW-TV labeled with *do*-leaders.

how to minimize the total leader length with rectilinear leaders in this setting. Their algorithm runs in $O(n^2)$ time. Minimizing the number of bends was left as an open problem. In this paper we consider multiple natural optimization criteria (among them minimum number of bends), we consider leaders with smoother bends (using obtuse angles), and for the case of rectilinear leaders with minimum total length, we improve the running time to $O(n \log n)$.

Our methods for general optimization criteria can be generalized to the two-sided problem, but the result is not relevant in practice due to running times of $O(n^8)$ and $O(n^{14})$. However, when this work was under review, Bekos et al. [2] meanwhile extended the class of obtuse-angled leaders studied in this paper further and obtained $O(n^3)$-time algorithms for length minimization with labels on one, two, or four sides of the illustration.

We will now state our problem more precisely.

Problem statement. We are given a set P of n points and n disjoint rectangles called *labels*. Each label is assumed to be large enough to fit the description of any point of P. The positions of the labels are fixed and their right edges all lie on a common vertical line, which lies to the left of all points in P. No two labels touch each other. A variant of the problem considers labels on two sides of P: one set of labels that have their right edges on a vertical line left of the points in P, and one set of labels that have their left edges on a vertical line right of the points in P.

In the one-sided case, labels can be connected to points by *leaders* that consist of two line segments: a horizontal segment, called the *arm*, that is attached to the right edge of the label and extends to the right, and a second

segment, called the *hand*, that connects the arm to the point. In all leaders the angle between the arm and the hand must be $180° \pm \alpha$ for some constant $0° < \alpha \leq 90°$ also denoted as the *bend angle* of a leader. If $\alpha = 90°$ the leaders are called *po-leaders* (consisting of a *p*arallel and an *o*rthogonal segment with respect to the labeled side of the illustration [5]); if $0° < \alpha < 90°$, we call them *do-leaders* (consisting of a *d*iagonal and an *o*rthogonal segment with respect to the labeled side of the illustration[1]). Both leader types are illustrated in Figure 1. If the arm connects the label directly to the point, omitting a hand, the leader is a *direct leader*. When α is fixed, a leader l is fully specified by its point $p(l)$ and the height (y-coordinate) of its arm. We assume that the 'badness' of a leader l is given by a function $bad(l)$. Natural choices for $bad(l)$ would be, for example, the length of l or the number of bends (0 or 1), or functions taking the interference of leaders with the underlying map into account. A *labeling* L is a set of n leaders that connects every point to a unique label and every label to a unique point. If no two leaders in L intersect each other, we say that L is *crossing-free*.

The problem we want to solve is the following: for a given set of points, a given set of labels, a given angle α, and a given badness function $bad()$, find a crossing-free labeling L such that the total badness $\sum_{l \in L} bad(l)$ is minimized.

Note that a leader may be attached anywhere to the right edge of a label, not necessarily at a fixed, prescribed position. Thus our problem is to find an optimal labeling using so-called *sliding* leader attachment.

The only difference in the two-sided case is that the arms of the leaders can connect to labels to the left of the points in P or to labels to the right of the points in P. We will focus on the one-sided case if not stated otherwise and only briefly discuss the two-sided case.

Our results. In Section 2 we present algorithms for *po*-leaders ($\alpha = 90°$): an $O(n^3)$-time algorithm that works with arbitrary badness functions, and an $O(n \log n)$-time algorithm for labelings with minimum total leader length (thus improving the $O(n^2)$-bound of Bekos et al. [5]). In Section 3 we present algorithms for *do*-leaders ($0° < \alpha < 90°$): again first a general algorithm, which runs in $O(n^5)$ time, and then a faster algorithm for minimum total leader length, which takes $O(n^2)$ time. In Section 4 we briefly show how our algorithms for arbitrary badness functions can be generalized to the situation where labels are located on two opposite sides of the point set. However, the running times are fairly high: $O(n^8)$ for *po*-leaders and $O(n^{14})$ for *do*-leaders. In Section 5 we present results from a case study, and we briefly discuss possible extensions in Section 6.

2 One-sided boundary labeling using *po*-leaders

In this section we study how to compute an optimal crossing-free labeling with leaders that have bends at an angle of 90°. In Section 2.1 we describe a general

[1] Following the naming scheme of Bekos et al. [5].

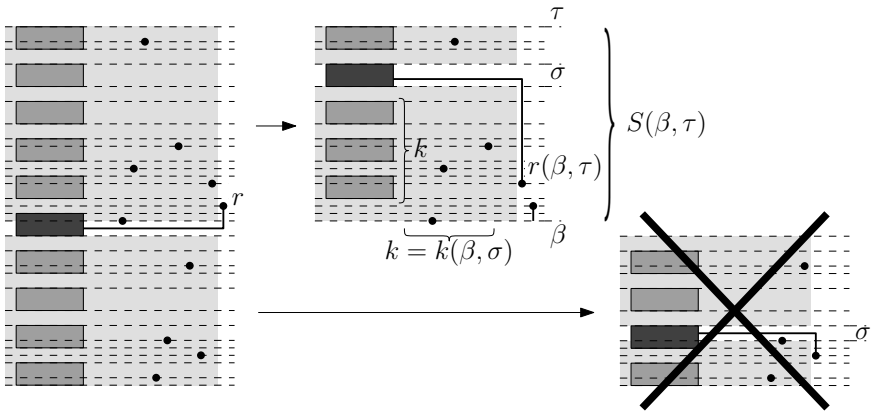


Figure 4: The recursive structure of an optimal solution. By the choice for the strip that contains the arm of the leader to the rightmost point, the problem is separated into two subproblems. As illustrated by strip σ in the lower subproblem, not all choices for the separating strip σ yield feasible subproblems: in this case there are two points and only one label below σ.

solution that works for any badness function $bad()$. In Section 2.2 we will give a faster solution for the case where $bad(l)$ is simply the length of l.

For simplicity we assume that none of the points to be labeled lies on a horizontal or a vertical line with another point, and no point lies on a horizontal line with the top or the bottom edge of a label. In Section 6 we briefly discuss what to do if these assumptions do not hold.

2.1 A dynamic programming algorithm for general badness functions

We present a dynamic programming solution based on the following idea. Let r be the rightmost point to be labeled. Consider any optimal crossing-free labeling L; let ℓ be the label associated with r in L. Then L consists of a leader l connecting ℓ to r, an optimal crossing-free labeling for the remaining labels and points below the arm of l, and an optimal crossing-free labeling for the remaining labels and points above the arm of l—see Figure 4.

Consider the subdivision of the plane into $O(n)$ strips, induced by the horizontal lines through the points and the horizontal edges of the labels. Note that the bottommost strip is unbounded in downward direction, and the topmost strip is unbounded in upward direction. To decide which labels and points lie below the leader l to r, we only need to know in which strip the arm of l lies; we do not need to know where exactly it is in the strip. When an arm lies on a strip boundary, we can consider it to lie in the strip above the boundary or in the strip below; the choice determines whether a point on the strip boundary is considered to lie above or below the leader.

Therefore, an optimal crossing-free labeling can be found by trying all possible choices of the strip σ in which to place the arm of the leader to r. For each choice, compute the optimal leader to r that has its arm in σ, and compute the optimal crossing-free labelings below and above the arm recursively. Note that we only need to consider *feasible choices* of σ, that is, choices of σ such that the number of labels and the number of points below σ and to the left of r are the same (for other choices of σ no labeling would be possible). In this case, as can be seen in Figure 4, the points to be matched below σ are simply the leftmost k points in the region defined by the strips below σ, where k is the number of labels below σ; analogously, the points to be labeled above σ are the leftmost points in the region defined by the strips above σ.

Let us denote by $S(\beta, \tau)$ the set of strips between strip β (bottom) and τ (top), excluding β and τ. Let $r(\beta, \tau)$ be the k-th leftmost point in $S(\beta, \tau)$, where k is the number of labels $k(\beta, \tau)$ that lie completely inside $S(\beta, \tau)$. Our recursive approach thus solves subproblems of the following form: for the set of strips $S(\beta, \tau)$, compute the optimal matching between the labels that lie completely inside $S(\beta, \tau)$ and the matching number of leftmost input points inside (and on the boundary of) $S(\beta, \tau)$. The minimum total badness $BAD[\beta, \tau]$ of the optimal crossing-free labeling for $S(\beta, \tau)$ is zero if $k(\beta, \tau) = 0$, and otherwise it can be expressed as:

$$BAD[\beta, \tau] = \min_{\text{feasible } \sigma \in S(\beta,\tau)} bad(l^*(r(\beta, \tau), \sigma)) + BAD[\beta, \sigma] + BAD[\sigma, \tau]$$

where $l^*(r(\beta, \tau), \sigma)$ is the optimal leader to $r(\beta, \tau)$ with its arm in strip σ.

Theorem 1 *Assume we are given a set of n points P, a set of labels on the left as described in Section 1, and a badness function $bad()$ such that we can determine, in $O(n)$ time, the badness and the location of an optimal po-leader to a given point with its arm in a given height interval (independent of the location of other leaders). We can compute a crossing-free labeling with po-leaders for P as defined in Section 1 with minimum total badness in $O(n^3)$ time and $O(n^2)$ space.*

Proof: We first sort all labels and points by y-coordinate, and all points by x-coordinate, which requires $O(n \log n)$ time. We also compute and store $l^*(p, \sigma)$ and $bad(l^*(p, \sigma))$ for every point p and every strip σ, in $O(n^3)$ time and $O(n^2)$ space. Then we compute the optimal crossing-free labeling by dynamic programming with memoization. Apart from the recursive calls, solving a subproblem requires deciding for which choices of σ the number of labels below σ matches the number of points below σ, and looking up $l^*(r(\beta, \tau), \sigma)$ and $bad(l^*(r(\beta, \tau), \sigma))$ for those strips. Given the list of all points sorted by x-coordinate and the list of labels and points by y-coordinate, we can construct a list of all labels and points in the given subproblem sorted by y-coordinate in $O(n)$ time. By scanning this list, we can determine in $O(n)$ time which choices of σ yield feasible subproblems. The number of different subproblems that need to be solved is quadratic in the number of strips, so we need to solve $O(n^2)$ subproblems which require $O(n)$ time each, taking $O(n^3)$ time in total. □

Corollary 1 *Assume we are given a set of n points P and a set of labels as described in Section 1. We can compute a crossing-free labeling with po-leaders for P as defined in Section 1 with the minimum total number of bends in $O(n^3)$ time and $O(n^2)$ space.*

2.2 A sweep-line algorithm for leader length minimization

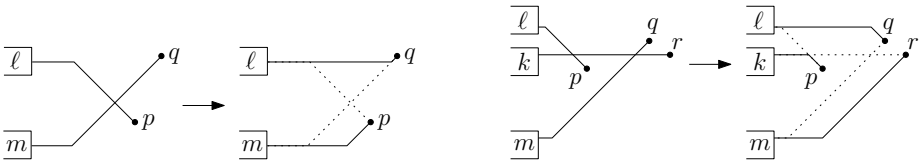
For the special case of minimizing the total leader length one can do better than in $O(n^3)$ time. We will give an algorithm that runs in $O(n \log n)$ time and show that this bound is tight in the worst case. However, before giving our algorithm, we first prove the following Lemma, which we need for the proof of correctness of our fast algorithms in this section and in Section 3.2.

Lemma 1 *For any labeling L^* with po- or do-leaders that may contain crossings and has minimum total leader length, there is a crossing-free labeling L whose total leader length does not exceed the total leader length of L^*. This labeling L can be constructed from L^* in $O(n^2)$ time.*

Proof: Let $l(p, \ell)$ denote the leader from point p to label ℓ. We say that $l(p, \ell)$ is an *upward leader* if the arm of the leader lies above p; the leader is a *downward leader* if the arm of the leader lies below p.

We first observe that L^* does not contain any crossings between an upward leader $l(p, \ell)$ and a downward leader $l(q, m)$: it is easy to see that if such a crossing exists, we could reduce the total leader length by replacing the leaders $l(p, \ell)$ and $l(q, m)$ by $l(p, m)$ and $l(q, \ell)$. Hence, L^* would not be optimal, see Figure 5a.

In a similar way we can verify that in L^* no direct leader can cross an upward and a downward leader at the same time: let $l(r, k)$ be a direct leader that crosses an upward leader and a downward leader; of the last two, let $l(p, \ell)$ be the leader that crosses $l(r, k)$ closest to k, and let $l(q, m)$ be the other leader crossing $l(r, k)$. We could now reduce the total leader length by replacing these leaders by $l(p, k)$, $l(q, \ell)$, and $l(r, m)$, see Figure 5b.



(a) If differently oriented leaders intersect, a labeling of smaller total leader length is possible.

(b) If a direct leader intersects differently oriented leaders, a labeling of smaller total leader length is possible.

Figure 5: Illustrations for the proof of Lemma 1.

We call a crossing that involves a downward leader a downward crossing, and a crossing that involves an upward leader an upward crossing. From the above

observation it follows that the set of leaders involved in downward crossings contains downward leaders and possibly direct leaders, the set of leaders involved in upward crossings contains upward leaders and possibly direct leaders, and no leader appears in both sets.

Below we explain how L^* can be transformed into a labeling without upward crossings without increasing the total leader length and without introducing more downward crossings. The transformation can be carried out in $O(n^2)$ time. By repeating the transformation upside-down, we can subsequently eliminate all downward crossings without re-introducing upward crossings, thus obtaining a crossing-free labeling with the same total leader length as L^*.

(a) before processing ℓ_i (b) after processing ℓ_i

Figure 6: Purging upward crossings.

To eliminate the upward crossings we proceed as follows. The approach follows the idea of Bekos et al. [5], generalized to leaders with any bend angle α (we will need this generalization in the next section of our paper). Let $\ell_1, \ldots, \ell_n$ be the sequence of all labels ordered from bottom to top. We process the labels in this order and make sure that all leaders ending at already processed labels are not involved in upward crossings anymore. Now assume that we are about to process label ℓ_i and its leader $l(p, \ell_i)$. In $O(n)$ time we determine the leader $l(q, \ell_j)$ that is involved in the leftmost upward crossing with $l(p, \ell_i)$—if there is any. Since all labels below ℓ_i have already been processed and are not involved in any upward crossings anymore, ℓ_j must lie above ℓ_i. This implies that the crossing is located on the arm of $l(p, \ell_i)$, see Figure 6a. We now swap the label assignment and replace $l(p, \ell_i)$ and $l(q, \ell_j)$ by $l(p, \ell_j)$ and $l(q, \ell_i)$, see Figure 6b. Obviously this does not change the total leader length. Both new leaders $l(p, \ell_j)$ and $l(q, \ell_i)$ are upward leaders and hence they cannot be involved in any new downward crossings—otherwise the new labeling, and therefore also the original labeling L^*, would be suboptimal. Regarding upward crossings, the hand of $l(q, \ell_i)$ is crossing-free since all leaders to labels below ℓ_i are crossing-free, and the arm of $l(q, \ell_i)$ is crossing-free by construction. Hence, $l(q, \ell_i)$, the leader to ℓ_i, is not involved in any upward crossing and we can proceed with ℓ_{i+1}.

Since each of the n labels is processed exactly once, using $O(n)$ time per label, the total running time is $O(n^2)$. □

Now we describe our $O(n \log n)$-time algorithm to compute a crossing-free labeling with *po*-leaders of minimum total length. The algorithm first scans the input to divide it into parts that can be handled independently; then it uses a sweep-line algorithm for each of these parts.

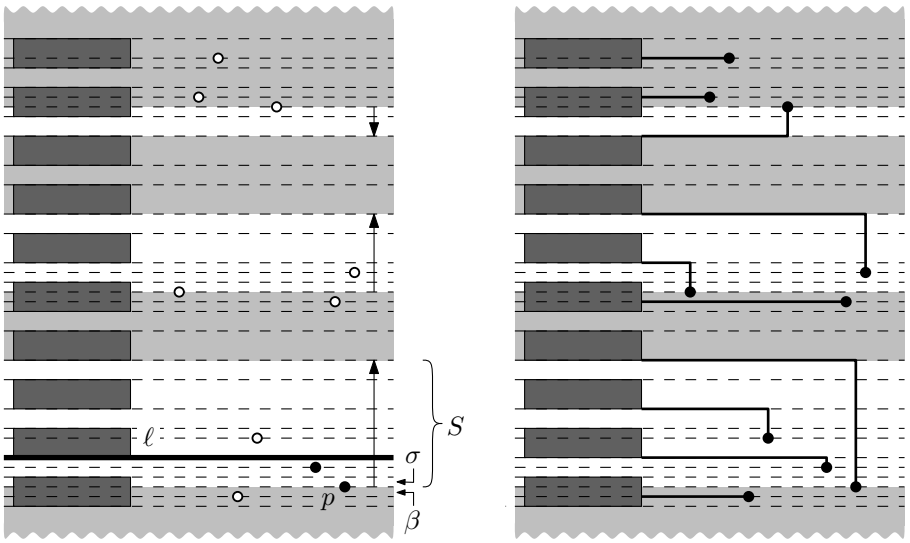


Figure 7: Left: Classification of strips in the plane sweep algorithm: neutral strips are shaded, downward and upward strips are marked by arrows. When the sweep line reaches the label ℓ, the two black points are in W. Right: The completed minimum-length labeling.

The initial scan works as follows. Consider the horizontal strips defined in the previous subsection. We traverse these strips in order from bottom to top, counting for each strip σ:

- pa_σ: number of points above σ (incl. any point on the top edge of σ);
- ℓa_σ: number of labels above σ (incl. any label intersecting σ);
- pb_σ: number of points below σ (incl. any point on the bottom edge of σ);
- ℓb_σ: number of labels below σ (incl. any label intersecting σ).

Note that for every strip σ, $pa_\sigma + pb_\sigma = n$, and $\ell a_\sigma + \ell b_\sigma$ is either n (when σ does not intersect any label) or $n+1$ (when σ intersects a label). We classify the strips in three categories and then divide the input into maximal sets of consecutive strips of the same category (see Figure 7):

- *downward*: strips σ such that $pa_\sigma > \ell a_\sigma$ (and therefore $pa_\sigma - \ell a_\sigma \geq 1$, $pb_\sigma < \ell b_\sigma$, and $pb_\sigma - \ell b_\sigma \leq -1$);
- *upward*: strips σ such that $pb_\sigma > \ell b_\sigma$ (and therefore $pb_\sigma - \ell b_\sigma \geq 1$, $pa_\sigma < \ell a_\sigma$, and $pa_\sigma - \ell a_\sigma \leq -1$);
- *neutral*: the remaining strips; these have $pa_\sigma = \ell a_\sigma$ and/or $pb_\sigma = \ell b_\sigma$.

Neutral sets are handled as follows: any point p that lies in the interior of a neutral set is labeled with a direct leader.

Points in an upward set S (including any points on its boundary) are labeled as follows. We use a plane sweep algorithm, maintaining a waiting list W of points to be labeled, sorted by increasing x-coordinate. Initially W is empty. We sweep S with a horizontal line from bottom to top. During the sweep two types of events are encountered: *point events* (the line hits a point p) and *label events* (the line hits the bottom edge of a label ℓ). When a point event happens, we insert the point in W. When a label event happens, we remove the leftmost point from W and connect it to ℓ with the shortest possible leader. (Assigning the leftmost point to ℓ prevents producing crossings in the further run of our algorithm.)

Points in downward sets are labeled by a symmetric plane sweep algorithm, going from top to bottom.

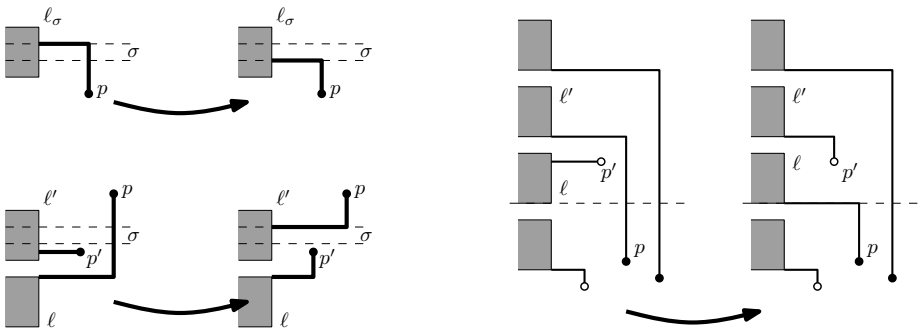
Lemma 2 *Given a set of n points P and a set of n labels on the left as described in Section 1, we can compute a crossing-free labeling with po-leaders as defined in Section 1 with minimum total length in $O(n \log n)$ time and $O(n)$ space in the worst case.*

Proof: The algorithm described above can easily be implemented to run in $O(n \log n)$ time and $O(n)$ space. We will now prove the correctness, that is, we show that the algorithm indeed produces a crossing-free labeling of minimum total leader length.

We observe that in any optimal labeling L, no leader crosses a neutral strip. To see this, consider any neutral strip σ. Assume $pa_\sigma = \ell a_\sigma$ (otherwise we have the symmetric case $pb_\sigma = \ell b_\sigma$). Let ℓ_σ be the label that intersects σ, if it exists. Suppose L contains a leader l that crosses σ. We consider three cases:

- If l connects a point p to ℓ_σ, then l can be shortened by moving its arm to the edge of σ closest to p, eliminating the intersection with σ (Figure 8a, top).
- If l leads from a point p above σ to a label ℓ below σ, then, because $pa_\sigma = \ell a_\sigma$, there must also be a leader l' leading from a point p' below σ to a label ℓ' above or intersecting σ. Now the total leader length can be reduced by connecting p to ℓ' and p' to ℓ (Figure 8a, bottom).
- The case that l leads from a point p below σ to a label above σ is symmetric to the previous case.

Swapping the labels of p and p' may cause leaders to intersect each other. Therefore the above argument only shows that if L contains a leader that crosses a neutral strip, then L is not optimal among all, not necessarily crossing-free labelings. However, Lemma 1 shows that for any optimal labeling that has crossings, there exists a crossing-free labeling that is equally good. Thus the above argument also shows that if L contains a leader that crosses a neutral strip, then L is not optimal among all crossing-free labelings.



(a) Eliminating intersections with neutral strips

(b) The black points are in W when the sweep line reaches ℓ.

Figure 8: Illustrations for the proof of Lemma 2

It follows that in any optimal labeling, points in the interior of neutral sets are labeled by direct leaders—as done by our algorithm. Observe that between any downward strip and any upward strip, both $pa_\sigma - \ell a_\sigma$ and $pb_\sigma - \ell b_\sigma$ differ by at least two. When going from a strip to an adjacent strip, the value of each of these expressions changes by at most one. Hence downward strips and upward strips are always separated by neutral strips. It follows that in any optimal labeling, the points in any upward (or downward) set S must be labeled by leaders that lie entirely within S. We will now argue that our plane sweep algorithm for such a set S produces an optimal labeling for the points in S.

Consider an upward set S. Note that the strip β directly below S is a neutral strip with $pb_\beta \leq \ell b_\beta$ while the bottommost strip σ in S has $pb_\sigma > \ell b_\sigma$; hence we have $pb_\beta = \ell b_\beta$ and the first event must be a point event for a point p on the bottom edge of S. Observe that the strips β and σ may intersect a label ℓ (as in Figure 7), but it cannot be used to label p: since $pb_\beta = \ell b_\beta$ and no leaders can cross β, all labels up to and including ℓ are needed to label points below β. So we must label all points in (and on the boundary of) S with labels that lie entirely above σ. It remains to prove that our algorithm produces such a labeling and that it is optimal.

First note that as soon as the number of label events processed catches up with the number of point events processed, we enter a neutral strip and the plane sweep of S ends; thus, whenever a label event occurs, W contains at least one point. Now consider a labeling L that deviates from the one produced by our algorithm. Let ℓ be the lowest label that, according to L, is *not* connected to the leftmost point p that is in W when the sweep line reaches ℓ. Note that it follows that p is connected to a label ℓ' above ℓ with a leader that crosses all strips that intersect ℓ; see Figure 8b. Now ℓ cannot be connected to any other point in W, because that would create a crossing with the leader between p and ℓ'. So ℓ must be connected to a point p' above the sweep line—but then swapping the labels of p and p' and subsequently resolving any resulting intersections would give a

labeling with smaller total leader length. Hence any labeling that deviates from the one produced by our algorithm must be suboptimal. □

The following lemma shows that this algorithm is best possible in an asymptotic sense.

Lemma 3 *The one-sided length minimization problem for po-leaders requires* $\Omega(n \log n)$ *to solve in the worst case.*

Proof: We prove the lemma by reduction from sorting. Consider a sequence $x_1, x_2, \ldots, x_n$ of distinct positive numbers, not necessarily in sorted order. Suppose we compute a crossing-free labeling with *po*-leaders for a set of points $(x_1, y_1), \ldots, (x_n, y_n)$, where $0 < y_i < 1$ for $1 \leq i \leq n$ and $y_i \neq y_j$ for $1 \leq i < j \leq n$, and a set of labels with lower right corners $(0, 1), (0, 2), ..., (0, n)$. Then we can look up the points attached to the labels at $(0, 1), (0, 2), ..., (0, n)$ in that order and will find the points sorted in order of increasing x-coordinate—otherwise leaders would intersect. Hence computing a crossing-free labeling is at least as difficult as sorting, which is known to take $\Omega(n \log n)$ time in the worst case. □

From Lemma 2 and Lemma 3 we get:

Theorem 2 *Given a set of* n *points* P *and a set of* n *labels on the left as described in Section 1, computing a crossing-free labeling with po-leaders as defined in Section 1 with minimum total length takes* $\Theta(n \log n)$ *time and* $\Theta(n)$ *space in the worst case.*

3 One-sided boundary labeling using *do*-leaders

In this section we study how to compute an optimal labeling with leaders that have a fixed bend angle $0° < \alpha < 90°$. First, in Section 3.1, we describe a general solution that works for any badness function $bad()$. Similar to the case of *po*-leaders, we will give a faster algorithm in Section 3.2 for the case where $bad(l)$ is simply the length of l.

For simplicity we assume that no two points lie on a line that has an angle of $0°$, $90°$, or $\pm\alpha$ with the x-axis, and no point lies on a horizontal line with an edge of a label. In Section 6 we briefly discuss what to do if these assumptions do not hold.

3.1 A dynamic programming algorithm for general badness functions

We use the same approach as for *po*-leaders, solving subproblems of the form: for a given region R, label the k points with the k labels in that region, where R is bounded from above and below by two leaders, and R is bounded on the right by the vertical line through the leftmost of the two points connected to those leaders. In fact a *po*-subproblem was fully defined by specifying the strips β

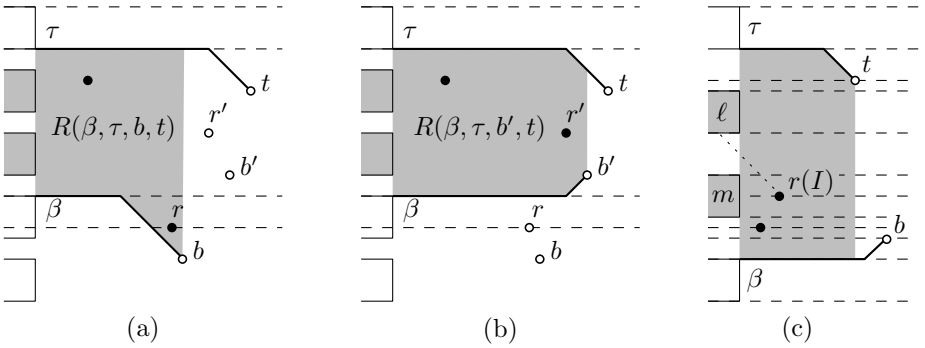


Figure 9: (a) The subproblem defined by β, τ, b, and t. (b) The subproblem defined by β, τ, b', and t. (c) Because of the fixed hand slope, $r(I)$ cannot have a leader with an arm in any strip: in the strip that contains ℓ, the hand would end up too far to the left; choosing a strip that intersects m would result in unbalanced subproblems; any other strip between β and τ does not intersect a label.

and τ that contain the arms of the leaders: this determined which labels lie inside R. However, with *do*-leaders we need to be more precise.

In addition to specifying β and τ we now also have to specify the points b and t to which the leaders that bound a subproblem are connected. This requirement is illustrated in Figures 9a and 9b: the subproblem defined by β, τ, b, and t contains the point r while the subproblem defined by β, τ, b', and t contains the point r' instead of r. The total number of different subproblems may thus increase to $O(n^4)$. We denote the region associated to the subproblem defined by β, τ, b, and t by $R(\beta, \tau, b, t)$, see Figure 9a. For simplicity we will abbreviate $I = (\beta, \tau, b, t)$ in the following.

Let $S(\beta, \tau)$ be the set of strips between strip β and τ, excluding β and τ. Let $r(I)$ be the rightmost point in $R(I)$. The minimum total badness $BAD[I]$ of the optimal crossing-free labeling is zero if there are no points in $R(I)$. Otherwise it can be expressed as:

$$BAD[I] = \min_{\text{feasible } \sigma \in S(\beta,\tau)} bad(l^*(r(I), \sigma)) + BAD[\beta, \sigma, b, r(I)] + BAD[\sigma, \tau, r(I), t],$$

where $l^*(r(I), \sigma)$ is the optimal leader to $r(I)$ with its arm in strip σ. Here again, a strip σ in $S(\beta, \tau)$ is feasible if the numbers of points and labels in the two emerging subproblems match.

An additional complication is that as a result of the fixed hand slope, not every subproblem with the correct number of labels and points can be solved (with *po*-leaders this was not a problem). Figure 9c shows a subproblem with two labels and two points that is infeasible since the two points in the problem together can reach only one label. This is detected as follows. A choice of σ in the subproblem defined by I is infeasible in each of the following situations:

- σ does not intersect a label;

- a hand attached to $r(I)$ would reach σ too far to the left to connect to the right edge of the label that intersects σ;
- a leader from $r(I)$ to the label that intersects σ would divide the problem into subproblems in which the numbers of points and labels do not match.

We define the minimum over an empty set to be ∞. If every choice of σ is infeasible, $BAD[I]$ thus evaluates to ∞, indicating that the problem defined by $I = (\beta, \tau, b, t)$ is infeasible.

The original input instance is infeasible if and only if every possible way of dividing the problem into subproblems leads to a subproblem in which no label can be matched to the rightmost point. Thus, if the original input instance is infeasible, the dynamic programming algorithm will evaluate its badness function to ∞.

By similar arguments as used in the proof of Theorem 1 we can identify the labels and points in a subproblem and all feasible choices of σ in $O(n)$ time per subproblem. We have $O(n^4)$ subproblems; thus we get:

Theorem 3 *Assume we are given a set of n points P, a set of n labels on the left as described in Section 1, a bend angle $0° < \alpha < 90°$, and a badness function $bad()$ such that we can determine, in $O(n)$ time, the badness and the location of an optimal do-leader to a given point with its arm in a given height interval (independent of the location of other leaders). If there is a crossing-free labeling for P with do-leaders with bend angle α as defined in Section 1, we can compute such a labeling with minimum total badness in $O(n^5)$ time and $O(n^4)$ space. If such a labeling does not exist, we can report infeasibility within the same time and space bounds.*

3.2 A sweep-line algorithm for leader length minimization

Like with *po*-leaders, we use a plane sweep algorithm instead of dynamic programming to improve the running time for the special case of minimizing the total leader length. In the description of our algorithm we distinguish *downward diagonals* (lines of negative slope that have an angle of $-\alpha$ with the x-axis) and *upward diagonals* (lines of positive slope that have an angle of α with the x-axis). For each label ℓ we define three regions in the plane (see Figure 11):

- $A(\ell)$ is the relatively open half plane *above* the *upward* diagonal through the *upper* right corner of ℓ;
- $B(\ell)$ is the relatively open half plane *below* the *downward* diagonal through the *lower* right corner of ℓ;
- $R(\ell)$ is the complement of $A(\ell) \cup B(\ell)$.

Note that a *do*-leader from a point p to a label ℓ is possible if and only if $p \in R(\ell)$.

The core of our approach is a recursive sweep-and-divide algorithm that takes as input a list of labels $\mathcal{L}$ and points P sorted in the order in which they

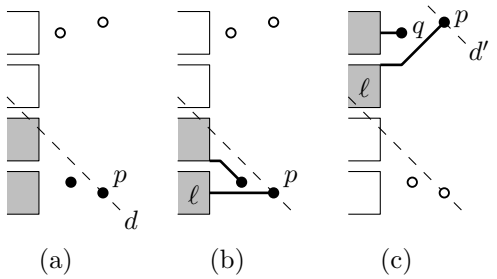


Figure 10: Illustration of the length-minimization algorithm for *do*-leaders. (a) When the sweep line d hits p, we make recursive calls on the input below the sweep line and the input above the sweep line. (b) The result of the recursive call below d. (c) The result of the recursive call above d. When the sweep line reaches d', point q is the lowest available point, but since q cannot reach ℓ, label ℓ is attached to p.

would be hit by a downward diagonal line that sweeps the plane bottom-up and from left to right. For any line d, let $\mathcal{L}(d)$ be the set of labels whose lower right corners lie below or on d, and let $P(d)$ be the set of points that lie below or on d. The algorithm sweeps the plane with a downward diagonal d up to the first point where we have $|P(d)| = |\mathcal{L}(d)|$. Observe that we will have to find a one-to-one matching between $P(d)$ and $\mathcal{L}(d)$, since no leaders are possible between points below d and labels above d. We find such a matching as follows.

If $P(d) \neq P$, we make a recursive call on $P(d)$ and $\mathcal{L}(d)$, and a recursive call on the remaining input $P \setminus P(d)$ and $\mathcal{L} \setminus \mathcal{L}(d)$, see Figure 10a.

If $P(d) = P$, we find the lowest label $\ell \in \mathcal{L}$. If no point of P lies in $R(\ell)$, we report that no labeling can be found and terminate the algorithm. Otherwise we draw a leader from ℓ to the lowest point p in $P \cap R(\ell)$ (as in Figures 10b and 10c); then, if $P \setminus \{p\}$ is not empty, we make a recursive call on $P \setminus \{p\}$ and $\mathcal{L} \setminus \{\ell\}$.

The full algorithm is now as follows. We first sort $\mathcal{L}$ and P into the order described above. Next, we run the recursive sweep-and-divide algorithm. If the algorithm does not fail, the computed set of leaders has minimum total length (as we will prove below), but it may contain crossings. We eliminate these crossings using Lemma 1.

Theorem 4 *Assume we are given a set of n points P, a set of n labels on the left as described in Section 1, and a bend angle $0° < \alpha < 90°$. If there is a crossing-free labeling for P with do-leaders with bend angle α as defined in Section 1, we can compute such a labeling of minimum total leader length in $O(n^2)$ time and $O(n)$ space in the worst case. If such a labeling does not exist, we can report infeasibility within the same time and space bounds.*

Proof: Initial sorting takes $O(n \log n)$ time. Each recursive call takes $O(n)$ time, excluding the recursive calls made from it. Since each recursive call is the

result of splitting the input into two non-empty parts, there are only $O(n)$ recursive calls, taking $O(n^2)$ time in total. By Lemma 1, eliminating intersections takes again $O(n^2)$ time, so the total running time is $O(n^2)$. Since the (at most) two recursive calls made from any call on the algorithm work on disjoint input lists, we can divide the input list of points and labels among the recursive calls without actually copying the input. Thus $O(n)$ space suffices.

It remains to prove that our algorithm produces a crossing-free labeling of minimum total leader length if one exists, and reports infeasibility only if no such labeling exists. We will show the following:

Claim 1 *Any (not necessarily intersection-free) labeling can be transformed into the labeling constructed by our recursive algorithm (before eliminating intersections) without increasing the total leader length.*

From this it follows immediately that our recursive algorithm produces a labeling of minimum total leader length if one exists. Lemma 1 guarantees that we can subsequently eliminate any intersections among the leaders while maintaining minimum total length. Furthermore, if no labeling exists, our algorithm will indeed report infeasibility, since our algorithm cannot terminate successfully without assigning every label ℓ to a point $p \in R(\ell)$ and never assigns more than one label to a point. We will now prove Claim 1.

Suppose we have a labeling L^* that deviates from the set of leaders L that is constructed by our algorithm. We will explain how we can transform L^* into L without increasing the total leader length. For any label ℓ, let $p^*(\ell)$ be the point connected to ℓ in L^*, and for any point p, let $\ell^*(p)$ be the label connected to p in L^*. Let $p(\ell)$, if it exists, be the point connected to ℓ in L. Now let ℓ be the lowest label such that our algorithm did *not* construct a leader between ℓ and $p^*(\ell)$. In other words, ℓ is the lowest label so that either our algorithm assigned ℓ to a point $p(\ell)$ that is different from $p^*(\ell)$, or our algorithm failed to assign a point to ℓ (and therefore reported infeasibility).

We will prove:

(i) that $p^*(\ell)$ must have been part of each recursive call that contains ℓ, so that our algorithm cannot have failed to assign a point $p(\ell)$ to ℓ;

(ii) that the label $m := \ell^*(p(\ell))$ which labels $p(\ell)$ in L^* lies above ℓ;

(iii) that we can change L^* such that the total leader length does not increase, all labels below ℓ remain connected to the same points, and $p(\ell)$ is reconnected to a label that lies lower than its current label m and thus closer to ℓ.

Repeating this transformation will eventually connect $p(\ell)$ to ℓ; repeating these transformations of L^* further we will make sure, going through all labels ℓ in bottom-up order, that L^* has the same leader as L for each label ℓ. This also shows that our algorithm succeeded in connecting a leader to every label and thus terminated successfully.

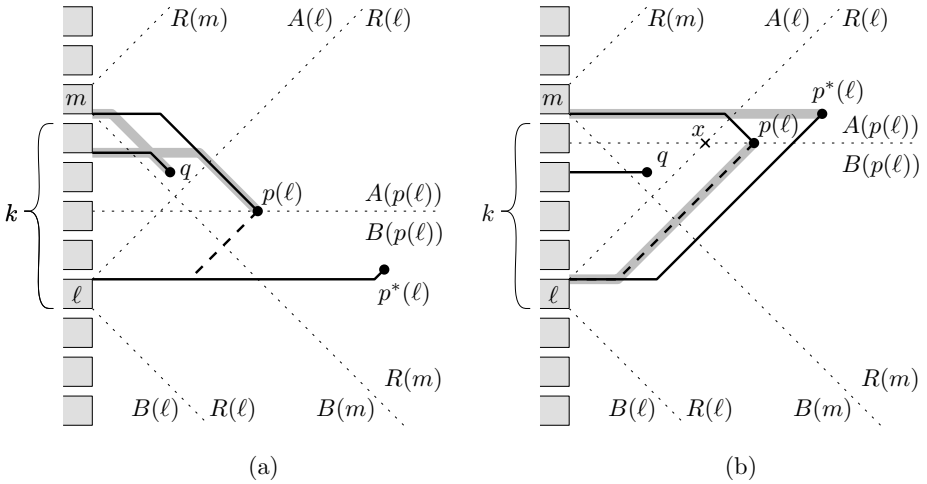


Figure 11: Illustration for claim (iii) of the proof of Theorem 4. Solid black lines are leaders in L^*; dashed black lines are leaders in L; grey lines are leaders after swapping. (a) q lies in $R(m) \cap A(p(\ell))$ and can swap its label with $p(\ell)$; (b) $p^*(\ell)$ lies in $R(m) \cap A(p(\ell))$ and can swap its label with $p(\ell)$.

(i) To prove the claim, assume, for the sake of contradiction, that there has been a call $\mathcal{C}$ in which ℓ and $p^*(\ell)$ were passed to different recursive calls. Then there must have been a downward diagonal d with the lower right corner of ℓ below d and $p^*(\ell)$ above d, such that in the input to $\mathcal{C}$ we have $|\mathcal{L}(d)| = |P(d)|$. Then, however, any labeling that is equal to L and L^* for all labels below ℓ, must match all labels in $\mathcal{L}(d)$ with points in $P(d)$, since leaders from the points in $P(d)$ cannot reach any other labels above ℓ. Specifically, L^* must connect ℓ to a point in $P(d)$, but this contradicts that $p^*(\ell)$ lies to the right of d. So the point $p^*(\ell)$ must have been part of each recursive call that includes ℓ. Hence, after successfully constructing leaders for each label below ℓ, our algorithm must have been able to construct a leader for ℓ: if not to $p^*(\ell)$, then to some other point $p(\ell)$.

(ii) By definition ℓ is the lowest label where L^* and L differ. Since $p^*(m) = p(\ell) \neq p(m)$ the two labelings L^* and L differ at m, which must therefore lie above ℓ.

(iii) For an illustration, refer to Figure 11. Let k be the number of labels between ℓ (inclusive) and m (exclusive). Let P be the set of input points not matched to labels below ℓ. Note that $p(\ell)$ lies in $R(m)$. The number of points in $P \cap B(m)$ is *less* than k, otherwise, before assigning a point to ℓ, our algorithm would divide ℓ and any points in $R(m)$ into different recursive calls, and thus it would not connect ℓ to $p(\ell)$. So in L^*, at least one label between ℓ (inclusive) and m (exclusive) is matched to another point q (other than $p(\ell)$) that lies above $B(m)$, and thus, in $R(m)$. Note that it is possible that $q = p^*(\ell)$.

Obviously $p(\ell)$ was also part of each recursive call that included ℓ; hence all labels above ℓ with lower right corners below the downward diagonal through $p(\ell)$ must have been part of each recursive call that included ℓ. These labels include all labels between ℓ and m inclusive. The argument under (i), that $p^*(\ell)$ must have been part of each recursive call that included ℓ, also holds for q and its label in L^*. Thus q must have been part of each recursive call that included the labels between ℓ and m inclusive. Hence the input to the recursive call that assigned $p(\ell)$ to ℓ must have included $p^*(\ell)$ and q.

Let $A(p(\ell))$ and $B(p(\ell))$ be the relatively open half planes above and below, respectively, a horizontal line through $p(\ell)$. We now argue that at least one point r out of $p^*(\ell)$ and q lies in $R(m) \cap A(p(\ell))$. First observe that q lies in $R(m)$ by definition. So if q lies in $A(p(\ell))$, as in Figure 11a, we are done. Now suppose q lies in $B(p(\ell))$, as in Figure 11b. Then, since the algorithm chose $p(\ell)$ for ℓ, not q, we must conclude that q lies outside $R(\ell)$, more precisely, q must lie in $A(\ell)$. So q lies in $R(m) \cap B(p(\ell)) \cap A(\ell)$. This implies that the intersection between the boundaries of $B(p(\ell))$ and $A(\ell)$ (marked by x in Figure 11b) lies to the right of the lower boundary of $R(m)$. Thus $A(p(\ell)) \cap R(\ell)$ lies in $R(m)$. Both $p(\ell)$ and $p^*(\ell)$ lie in $R(\ell)$, but the algorithm chose $p(\ell)$ for ℓ, not $p^*(\ell)$, so $p^*(\ell)$ must lie in $A(p(\ell)) \cap R(\ell)$, and thus, in $R(m)$. So at least one point r out of $p^*(\ell)$ and q lies in $R(m) \cap A(p(\ell))$.

Now observe that r lies above $p(\ell)$, while $\ell^*(r)$ lies below $\ell^*(p(\ell)) = m$. Furthermore r lies in $R(m)$, and $p(\ell)$ lies in $R(\ell^*(r))$, since $p(\ell)$ can reach any label between ℓ and m. Hence it is possible to swap the labels of $p(\ell)$ and r in L^*. This will not increase the total leader length; $p(\ell)$ gets assigned a lower label than before, while the leaders to labels below ℓ do not change.

This proves Claim 1, which also concludes the proof of Theorem 4. □

4 Two-sided boundary labeling

The dynamic-programming approaches of the previous sections can be generalized to the setting where labels are placed on two opposite sides of the rectangular region that contains the input points. The running times of our two-sided algorithms are $O(n^8)$ for *po*-leaders and $O(n^{14})$ for *do*-leaders. Unfortunately, the algorithms are of purely theoretical interest since it turned out that they perform as bad in practice as the theoretical bounds suggest. Even for instances with 10 points it already took several minutes to compute an optimal labeling. We will focus on sketching the ideas that generalize the algorithms for the one-sided case; we omit a complete exposition of the technical details, which do not bear any substantially new ideas compared to the one-sided cases.

The idea for using dynamic programming to tackle the two-sided problem is the same for both leader types: we partition the region of a two-sided problem into two subregions by a polygonal splitting line g, trying several possibilities. Each of the two subregions with its enclosed points and adjacent labels represents a new subproblem. These subproblems can either be one-sided, in which case we can basically use the one-sided algorithms of Section 2.1 and Section 3.1

to solve them, or they remain two-sided, in which case we use another polygonal splitting line to partition them further.

The emerging subproblems can be solved independently since we postulate that we only need to compute labelings whose leaders do not intersect g. To show the correctness of the dynamic programming algorithm we have to prove that for any two-sided input instance and a fixed optimal labeling L, there exists a splitting line g such that no leader of L intersects g, and that the algorithm will indeed try this splitting line g in its search for an optimal solution.

4.1 A dynamic programming algorithm for *po*-leaders

First, we characterize the splitting lines g that can be used to partition a two-sided instance I that is to be labeled using *po*-leaders.

Let L be a fixed optimal labeling for I with respect to some given badness function $bad()$. Let v be a vertical line that passes between the labels on the left and the labels on the right, and splits the points such that the number of points on either side of v matches the number of labels, see Figures 12a and 12b.

If no leader of L intersects v, then L (or another optimal solution) can be found by solving the two one-sided subproblems that result from splitting I at $g := v$, see Figure 12a.

If there are leaders in L that intersect v, the definition of a suitable splitting line g is more involved. Observe that there must be a balance between the leaders of L that intersect v: the number of leaders labeling a point to the left of v with a label to the right equals the number of leaders labeling a point to the right of v with a label on the left. Traversing v from top to bottom, there must be at least one pair of subsequent intersections whose leaders are connected to labels on different sides, see Figure 12b. Let l_{left} and l_{right} be a pair of such leaders, leading to a label on the left and on the right, respectively (if there are more such pairs, we just take any of them). We will now define g. If the arm of l_{left} lies above the arm of l_{right} (as in Figure 12b), then g consists of a horizontal segment from the left towards v, just below the arm of l_{left}, a vertical segment v' on v between l_{left} and l_{right}, and a horizontal segment from v to the right, just above the arm of l_{right}. If the arm l_{left} lies below the arm of l_{right}, the splitting line g is defined analogously, exchanging above and below. By construction, no leader of L intersects g: no leader of L intersects the arm of any other leader, and because the intersections with l_{left} and l_{right} are consecutive on v, no leader intersects v'. Therefore L (or another optimal solution) can be found by solving the two subproblems that result from splitting I along g. Care should be taken with the labels intersected by g (if any): if the left horizontal segment of g lies above the right horizontal segment of g (as in Figure 12b), then the left label intersected by g belongs to the upper subproblem and the right label intersected by g belongs to the lower subproblem; if the left horizontal segment lies below the right horizontal segment, then it is the other way around.

We thus get the following: an optimal labeling L can be found by trying all possible splitting lines g that result in subproblems with matching numbers of points and labels; for each such g we solve the resulting subproblems recursively.

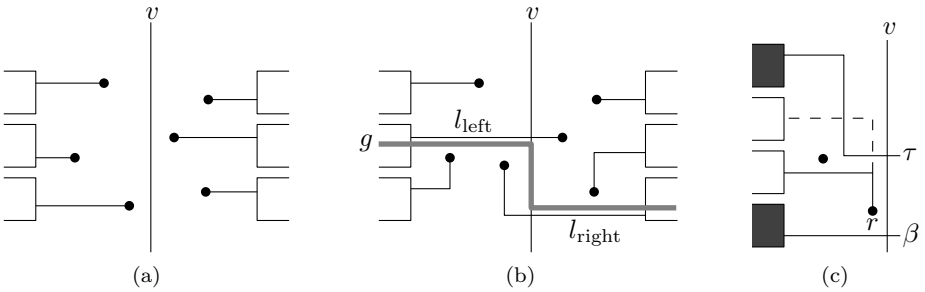


Figure 12: Splitting a two-sided *po*-leader problem into two subproblems. (a) no leader in L intersects v, (b) there are leaders intersecting v, (c) a left-sided subproblem. In (b) the polygonal splitting line g that respects L is displayed in bold grey. In (c) the dark labels do not belong to the instance defined by β and τ; the dashed leader for point r is infeasible as it intersects τ.

For the number of different possible splitting lines, consider the subdivision of the plane into $O(n)$ horizontal strips, induced by the horizontal lines through the points and the horizontal edges of the labels like in Section 2.1. The left and the right horizontal segment of g are defined by the strips in whose interiors they lie (as before, the exact position inside the strip does not matter). Note that according to our definition of g via two oppositely directed leaders intersecting the vertical line v it suffices to consider splitting lines whose left and right horizontal strips are distinct. Thus the assignment of the labels intersected by g to the two induced subproblems is well-defined. For the vertical segment of g consider the subdivision of the plane into $O(n)$ vertical strips induced by the vertical lines through the points and the vertical edges of the labels. Then the vertical segment is defined by the vertical strip in whose interior it lies. This amounts to $O(n)$ possibilities for each segment, and thus, to $O(n^3)$ possible polygonal splitting lines g.

Each subproblem is defined by a lower polygonal line β and an upper polygonal line τ consisting of three segments each as described above. Such a subproblem can be left-sided (it consists of the labels on the left side between β and τ, and the matching number of leftmost points between β and τ), right-sided (it consists of the labels on the right side between β and τ, and the matching number of righmost points between β and τ), or two-sided (it consists of all labels and points between β and τ). Thus the table size for the dynamic programming algorithm is $O(n^3) \cdot O(n^3) \cdot 3 = O(n^6)$.

The minimum total badness $BAD[\beta, \tau, x]$ of the optimal crossing-free labeling for the x-sided problem defined by β and τ is zero if $BAD[\beta, \tau, x]$ contains no points; otherwise the recursion is:

$$BAD[\beta, \tau, \text{two}] = \min \begin{cases} BAD[\beta, \tau, \text{left}] + BAD[\beta, \tau, \text{right}] \\ \min_{\text{feasible } g} BAD[\beta, g, \text{two}] + BAD[g, \tau, \text{two}] \end{cases}$$

and

$$BAD[\beta, \tau, x] = \min_{\text{feasible } \sigma} bad(l^*(r(\beta, \tau, x), \sigma)) + BAD[\beta, g_\sigma, x] + BAD[g_\sigma, \tau, x],$$

where $x \in \{\text{left}, \text{right}\}$, σ is the strip used by the arm of $l^*(r(\beta, \tau, x), \sigma)$, and g_σ is the degenerate splitting line $(\sigma, -, -)$ for $x = \text{left}$ and $(-, -, \sigma)$ for $x = \text{right}$ using the notation (a, b, c) for the splitting line with left, vertical, and right segment in strips a, b, and c, respectively.

In order to compute a two-sided table entry the recursion suggests to examine $O(n^3)$ splitting lines g. However, it turns out that it suffices to examine $O(n^2)$ table entries instead. The reason is that we can fix the vertical strip of the line v that partitions the points in the given instance I into a left and a right side so that the number of points and labels on each side match. As we have observed in the beginning of this section, for any vertical line v that partitions the points in this way, there is always a feasible polygonal splitting line g that uses a segment of v as its vertical segment such that splitting I along g leads to an optimal solution. Thus we can fix such a line v and fix the vertical segment of g to lie in the strip of that line v; then it suffices to examine the restricted set of $O(n^2)$ splitting lines for the one minimizing the recursive expression.

The computation of the one-sided table entries is similar to the one-sided algorithm in Section 2.1. However, here we need to pay attention to degenerate and non-degenerate region boundaries. Whether a label intersected by a non-degenerate splitting line belongs to an instance is defined as for the two-sided case. A label intersected by a degenerate splitting line never belongs to that instance since it has already been assigned to a point in a previous step. Figure 12c shows an example, where β is degenerate and τ is non-degenerate. The points belonging to an instance are the k leftmost (or rightmost) points between the upper and lower splitting line, where k is the number of labels of that instance. Finally, note that for a strip σ to be feasible in an one-sided problem the leader $l^*(r(\beta, \tau, x), \sigma)$ must lie completely inside the region between β and τ, see Figure 12c. At most $O(n)$ strips are examined to compute a one-sided table entry and thus the full table of size $O(n^6)$ is computed in $O(n^8)$ time as stated in the following theorem.

Theorem 5 *Assume we are given a set of n points P, a set of n labels on the left and on the right as described in Section 1, and a badness function $bad()$ such that we can determine, in $O(n)$ time, the badness and the location of an optimal po-leader to a given point with its arm in a given height interval (independent of the location of other leaders). We can compute a two-sided crossing-free labeling with po-leaders for P as defined in Section 1 with minimum total badness in $O(n^8)$ time and $O(n^6)$ space.*

Note, however, that for the special case of leader length as the badness function, Bekos et al. [5] gave a dynamic-programming algorithm that computes an optimum labeling in quadratic time and space.

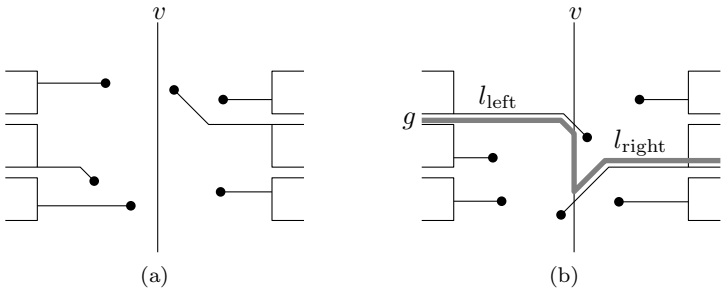


Figure 13: Splitting a two-sided *do*-leader problem into two subproblems. (a) no leader in L intersects v, (b) there are leaders intersecting v. The polygonal splitting line g that respects L is displayed in bold grey.

4.2 A dynamic programming algorithm for *do*-leaders

Now, we sketch how to partition a two-sided instance I that is to be labeled using *do*-leaders. We follow exactly the same argument as for the *po*-leaders. Again an instance of the problem can either be split into two one-sided subinstances by a vertical line v, or into two two-sided subinstances by a polyline g that first follows a leader on the left up to v, then follows part of v, and then follows a leader to a label on the right as depicted in Figure 13. The only difference is that in the case of *po*-leaders the first and the last part of g consist of a horizontal segment (on a leader arm), specified by the strip in which it lies, while in the case of *do*-leaders the first and last part of g may consist of a horizontal segment (an arm) *and* a diagonal segment (part of a hand), see Figure 13b.

In general there are $O(n^5)$ different splitting lines specified by two horizontal strips in which the arms lie, two points to which the hands lead, and a vertical strip connecting the left and the right arm and hand.

As each subinstance itself is defined by a lower and an upper polygonal line β and τ with five segments the table size for the dynamic programming is $O(n^{10})$. For the computation of a new table entry the same recursion as for two-sided *po*-leaders in the previous section holds. In order to compute a new two-sided table entry the recursion needs to find the minimum over a set of $O(n^5)$ possible splitting lines. As before, we can reduce the set of splitting lines to consider to a set of $O(n^4)$ candidates as follows. If we choose the vertical line v to be a line that balances the number of points and labels on each side of v this uniquely defines a vertical strip. As for *po*-leaders there is always a splitting line, which uses v as its vertical segment, that leads to an optimal solution. Hence we fix the vertical segment of g to lie in the strip of v and restrict our attention to the remaining $O(n^4)$ splitting lines. The computation of a one-sided entry is similar to the algorithm in Section 3.1 and takes into account only a linear number of table entries as before. This yields an overall running time of $O(n^{14})$ to compute the full table.

Theorem 6 *Assume we are given a set of n points P, a set of n labels on the left and on the right as described in Section 1, a bend angle $0° < \alpha < 90°$, and a badness function bad() such that we can determine, in $O(n)$ time, the badness and the location of an optimal do-leader to a given point with its arm in a given height interval (independent of the location of other leaders). If there is a crossing-free labeling for P with do-leaders with bend angle α as defined in Section 1, we can compute such a labeling with minimum total badness in $O(n^{14})$ time and $O(n^{10})$ space. If such a labeling does not exist, we can report infeasibility within the same time and space bounds.*

5 Experimental evaluation

To evaluate the practical relevance of our methods and their results we implemented the one-sided dynamic-programming algorithms for both *po*-leaders and *do*-leaders, as they can deal with arbitrary badness functions including length and bend minimization. For comparison we also implemented the fast $O(n \log n)$ sweep-line algorithm for the length minimization of *po*-leaders. We refrained from further experimental investigation of the two-sided algorithms as we found that solving *po*-leader instances with only 10 points already took several minutes. Three badness functions were considered: length minimization, bend minimization and a hybrid method combining both objectives. The corresponding badness functions bad_{len}, bad_{bend}, and bad_{hyb} are defined as follows.

$$bad_{\text{len}}(l) = |l|, \tag{1}$$

$$bad_{\text{bend}}(l) = \begin{cases} 0 & \text{if } l \text{ is direct} \\ 1 & \text{otherwise} \end{cases}, \tag{2}$$

$$bad_{\text{hyb}}(l) = \frac{|\text{hand}(l)|}{|\text{arm}(l)|} + \lambda_{\text{bend}} bad_{\text{bend}}(l), \tag{3}$$

where $|\cdot|$ denotes the Euclidean length. Note that in bad_{hyb} we do not simply reuse bad_{len} but rather include the length ratio of the hand and the arm of a leader. This is motivated by the observation that a long hand on a short arm looks worse than a hand of the same length on a long arm. The parameter λ_{bend} is used to adjust the relative weight of bad_{bend}.

Furthermore, we implemented another badness term bad_{cls} that measures how close points in P lie to a leader l within a neighborhood strip $N_\gamma(l)$ of width γ around l. This term can be added to the previous badness functions to avoid that leaders pass by other points with too little clearance. It is defined as

$$bad_{\text{cls}}(l) = \lambda_{\text{cls}} \sum_{p \in N_\gamma(l)} \left(1 - \frac{d(p,l)}{\gamma}\right)^2, \tag{4}$$

where λ_{cls} is a weight parameter and $d(p,l)$ is the Euclidean distance between p and l. The more points there are in $N_\gamma(l)$ and the closer they are to l, the higher is $bad_{\text{cls}}(l)$.

We implemented our algorithms in Java[2] and tested them on a map showing the 21 mainland regions of France, see Figures 14 and 15. The labelings were computed under SuSE Linux 10.3 on an AMD Opteron 248 2.2 GHz system with 4GB main memory. In our examples the running-time differences for evaluating the three badness functions for a particular leader are negligible. Thus the running times of the dynamic programming algorithms are independent of the actual badness function and we do not need to distinguish between different badness functions in the discussion below.

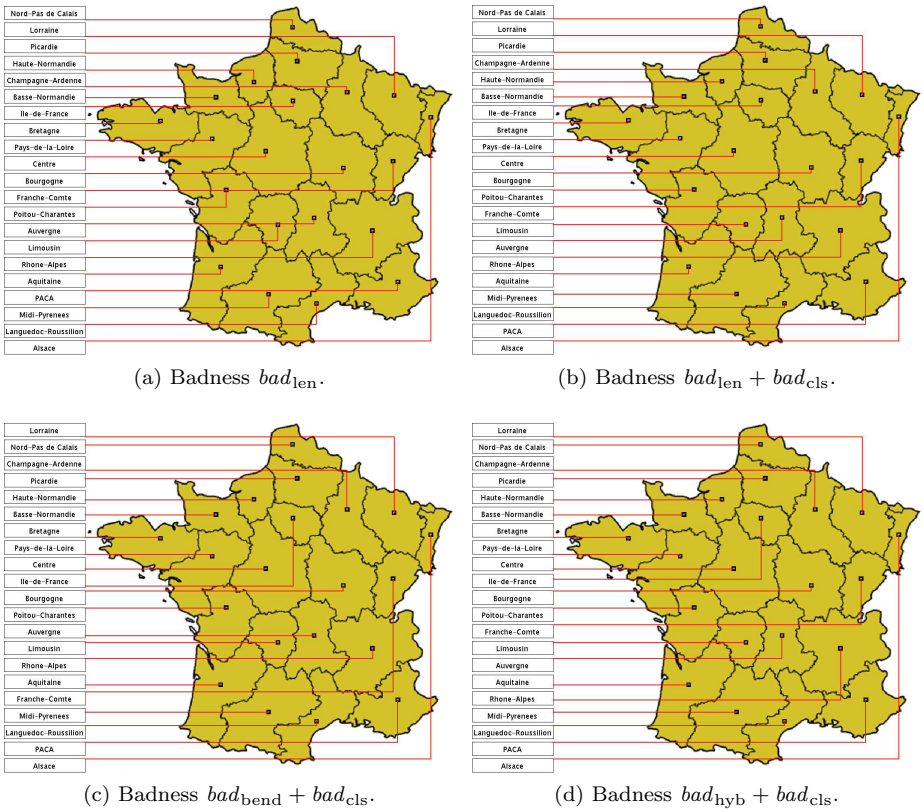
Averaged over 30 runs, the computations of the general algorithms took 2ms for *po*-leaders and 7ms for *do*-leaders with bend angle $\alpha = 45°$. The line-sweep algorithm for *po*-leaders took less than 1ms for this example. We ran the dynamic programming algorithms in a top-down fashion with memoization, so that only required entries in the dynamic programming table are computed. Thus 39% of $O(n^2)$ table entries were computed for *po*-leaders, while for *do*-leaders only 0.24% of $O(n^4)$ entries were computed. Since the feasibility of each subinstance is independent of the badness function, the same percentages were measured for all badness functions.

In practical labeling applications the maximum number of simultaneously labeled input points is usually in the order of a few dozen. So for getting a "practical" upper bound on the running times we tested the algorithms on artificially generated instances of $n = 100$ points uniformly distributed in a unit square. Here the computation of the *po*-leaders took about 0.3s averaged over 30 instances and on average 25% of the table entries were computed. The average running time for the *do*-leaders on the same instances was about 2.5s and on average 0.01% of the table entries were computed. This gap grows—as expected—such that for $n = 200$ input points the average running times were 1.3s for *po*-leaders and 17.1s for *do*-leaders. The table usage decreased to 14% for *po*-leaders and less than 0.002% for *do*-leaders.

For larger instances the fast $O(n \log n)$-time sweep-line algorithm shows its strength. The running times for the above instances with 100 and 200 points were still below 1ms. We increased n to 3,200, 6,400, and 12,800 points and measured average running times of 3ms, 8ms, and 19ms. This indicates that for large instances the line-sweep algorithm is far superior to the dynamic programming algorithms, as long as we use leader length as the badness function.

With short running times in the above range for the practically relevant small to medium-sized instances the visual quality of the labeling becomes the deciding criterion rather than computational performance. Hence we can afford using the (slower) dynamic programming algorithm, which gives us the flexibility to use more sophisticated badness functions. From the comparison of Figures 14a and 14b (see, for example, the leaders of Auvergne and Limousin) it is apparent that using the closeness penalty bad_{cls} in the badness function successfully avoids unwanted proximity of leaders, which could cause confusion when understanding the assignment of points and labels. The same observation can be made for *do*-leaders in Figures 15a and 15b. Apart from that, the decision of which labeling

[2] An applet is available at `http://i11www.iti.uni-karlsruhe.de/labeling`.



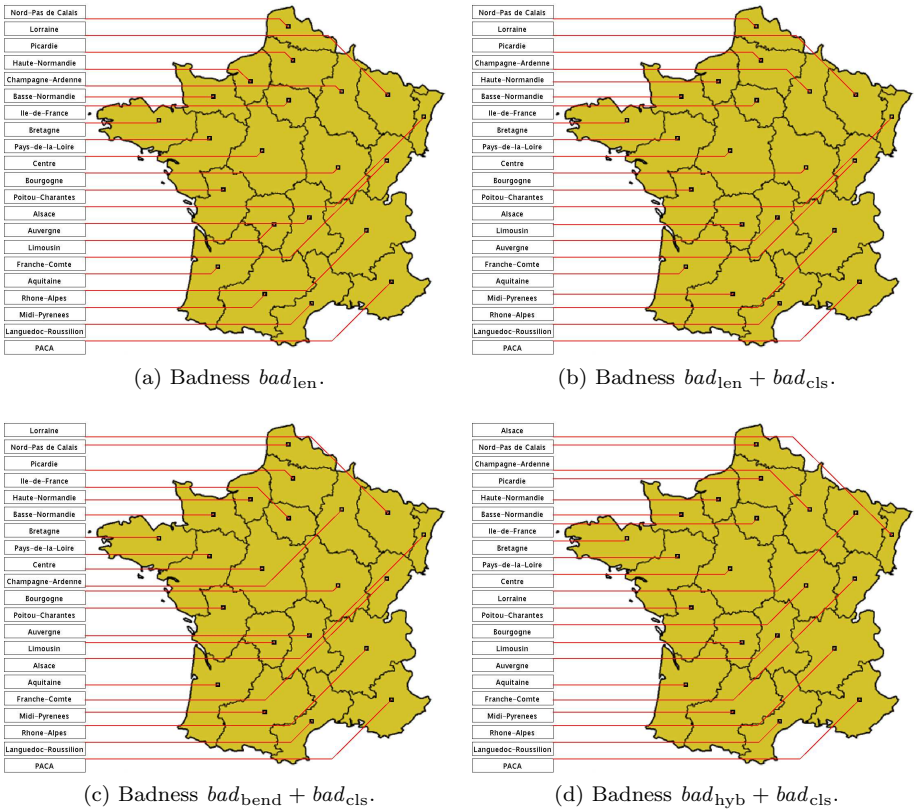
(a) Badness bad_{len}. (b) Badness $bad_{\text{len}} + bad_{\text{cls}}$.

(c) Badness $bad_{\text{bend}} + bad_{\text{cls}}$. (d) Badness $bad_{\text{hyb}} + bad_{\text{cls}}$.

Figure 14: One-sided *po*-labelings for the mainland regions of France.

is most suitable will ultimately be a matter of taste. In the following we give a brief discussion on the two types of leaders and the badness functions that we used.

***po*-leaders versus *do*-leaders.** Both types of leaders in Figures 14 and 15 have advantages and disadvantages. Obviously, it is not possible to judge whether *po*-leaders or *do*-leaders are generally superior based on our single example map. The answer depends both on the labeled image and on personal taste. Still, an advantage of the *do*-leaders is that due to the smoother angle their shape is easier to follow visually, which simplifies finding the correct label for a point and vice versa. On the other hand *po*-leaders add only line segments of two orientations to the background image, namely horizontal and vertical ones, while *do*-leaders add line segments of three orientations: horizontal and diagonal ones with positive and negative slope. Thus *po*-labelings might be less distracting when studying the illustration itself and not following leaders. Moreover, as seen in Section 3, a labeling with *do*-leaders is not always feasible.



(a) Badness bad_{len}.

(b) Badness $bad_{\text{len}} + bad_{\text{cls}}$.

(c) Badness $bad_{\text{bend}} + bad_{\text{cls}}$.

(d) Badness $bad_{\text{hyb}} + bad_{\text{cls}}$.

Figure 15: One-sided *do*-labelings for the mainland regions of France.

A possible solution was studied recently by Bekos et al. [2], who allow leaders with horizontal or vertical hands and diagonal arms (*od*-leaders and *pd*-leaders) so that a labeling is always possible.

Length minimization versus bend minimization. Minimizing the total leader length seems to give more comprehensible and visually more pleasing results than minimizing the total number of bends. One reason for this is that minimizing the length favors having each label close to the point being labeled. This results in a label assignment where the vertical order of the labels tends to reflect the vertical order of the points in the figure fairly well. In contrast, when minimizing the number of bends this correspondence is more easily lost, which can be confusing, compare Figures 14b and 14c as well as Figures 15b and 15c. In addition, the longer the hand segments are, the harder they are to follow and this is not considered in bad_{bend}. However, although direct leaders are easiest to read, their number should not be maximized without considering the shape and length of the non-direct leaders. Therefore the hybrid badness

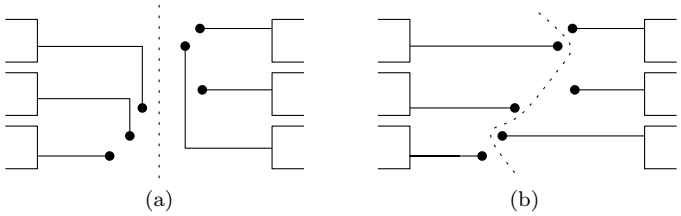


Figure 16: Splitting in the middle (a) to solve two one-sided problems is not necessarily the best choice (b), for example in terms of the number of bends.

function applied in Figures 14d and 15d is designed to find a good compromise between both optimization goals.

Discussion. We find that minimizing the length is more important for the aesthetic quality of a labeling than minimizing the bends. Combining both aspects in a hybrid badness function leads to a good compromise between the two objectives that has more direct leaders than the length-minimal solution. Furthermore the closeness term bad_{cls} turned out to be of great importance for good labelings.

6 Concluding remarks

In this paper we have presented efficient algorithms for solving the one-sided boundary labeling problem for both *po*-leaders and *do*-leaders. The algorithms either minimize the total leader length or optimize a general badness function. The evaluation of the implementation of our algorithms with different badness functions shows their potential for practical applications.

For two-sided boundary labeling our dynamic programming algorithms are not feasible in practice. For *po*-leaders, one can use the $O(n^2)$-time length-minimization algorithm of Bekos et al. [5]. For combinations of several types of obtuse-angled leaders one can use the $O(n^3)$-time length-minimization algorithm of Bekos et al. [2]. Alternatively—when it comes to general badness funtions—we suggest splitting the instance into two one-sided problems. We leave it as an open problem to find efficient algorithms for dividing points between the left and the right side in a way that yields good two-sided *po*- and *do*-labelings. Splitting in the middle does not necessarily yield aesthetically good results, see Figure 16. For *do*-leaders a feasible instance can even become infeasible by splitting in the middle.

For simplicity we assumed that no two points lie on a line that has an angle of $0°$, $90°$, or $\pm\alpha$ with the x-axis, and no point lies on a horizontal line with an edge of a label. If these assumptions do not hold, we can make them hold by perturbing the input slightly. However, this may allow leaders to pass very close to a point that would really be hit by the leader if it were not for the perturbation. Infeasible labeling problems may thus appear to be feasible when

they really are not. This problem may effectively be prevented by choosing an appropriate leader badness function that penalizes leaders that pass too close by points, as explained in Section 5.

An interesting future task is to reflect the interference of a leader and the background image in the badness function. Our dynamic programming algorithms can still be applied in this setting as long as the badness of a leader is independent of the other leaders.

In some applications it is desired to place the labels along the contour of the graphic rather than vertically aligned to the left of the bounding box. We can modify our dynamic programming algorithms such that they cover this situation as well.

If labels contain a detailed textual description and not just a name it is worth looking at the more general problem where each label has its own height which is given by the number of text lines times a unit height. With some modifications our dynamic programming algorithms are able to deal with this variant in pseudo-polynomial time.

Another interesting variant of the problem is labeling line segments or (polygonal) regions rather than single points, see Bekos et al. [4]. In that case both leader ends are flexible to some degree. It seems hard to adapt our dynamic programming solutions to this situation and further research is required.

References

[1] K. Ali, K. Hartmann, and T. Strothotte. Label layout for interactive 3D illustrations. *Journal of WSCG*, 13:1–8, 2005.

[2] M. A. Bekos, M. Kaufmann, M. Nöllenburg, and A. Symvonis. Boundary labeling with octilinear leaders. In J. Gudmundsson, editor, *Proc. 11th Scandinavian Workshop on Algorithm Theory (SWAT'08)*, volume 5124 of *Lecture Notes in Comput. Sci.*, pages 234–245. Springer-Verlag, 2008.

[3] M. A. Bekos, M. Kaufmann, K. Potika, and A. Symvonis. Multi-stack boundary labeling problems. In S. Arun-Kumar and N. Garg, editors, *Proc. Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *Lecture Notes in Comput. Sci.*, pages 81–92. Springer-Verlag, 2006.

[4] M. A. Bekos, M. Kaufmann, K. Potika, and A. Symvonis. Polygon labelling of minimum leader length. In K. Misue, K. Sugiyama, and J. Tanaka, editors, *Proc. Asia Pacific Symp. on Inform. Visualisation (APVIS'06)*, volume 60 of *CRPIT*, pages 15–21, 2006.

[5] M. A. Bekos, M. Kaufmann, A. Symvonis, and A. Wolff. Boundary labeling: Models and efficient algorithms for rectangular maps. *Computational Geometry: Theory and Applications*, 36:215–236, 2007.

[6] J.-D. Fekete and C. Plaisant. Excentric labeling: Dynamic neighborhood labeling for data visualization. In *Proc. of the SIGCHI conference on Human factors in computing systems (CHI'99)*, pages 512–519, 1999.

[7] H. Gray. *Anatomy of the Human Body.* Lea & Febiger, Philadelphia, 1918.

[8] A. Wolff and T. Strijk. The map-labeling bibliography. `http://i11www.iti.uka.de/~awolff/map-labeling/bibliography/`, 2006.