Algorithms for Multi-Criteria One-Sided Boundary Labeling*

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

[1] Faculty of Informatics, Karlsruhe University, P.O. Box 6980, 76128 Karlsruhe, Germany. http://i11www.iti.uka.de/algo/group
[2] Department of Computing Science, TU Eindhoven, Postbus 513, 5600 MB Eindhoven, Netherlands

Abstract. We present new algorithms for labeling a set P of n points in the plane with labels that are aligned to the left 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 algorithm finds a collection of non-intersecting leaders that minimizes the total number of bends, the total length, or any other 'badness' function of the leaders. An experimental evaluation of the performance is included.

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 [6]. However, this is not always possible: sometimes the labels are too large, the labeled features lie to 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 object by a curve—see 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 are examples where call-outs are used.

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

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

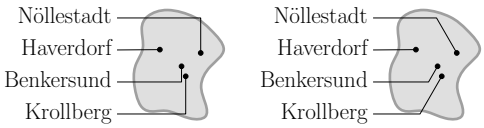


Fig. 1: Examples of call-outs with bends of 90° (*po*-leaders) or 120° (*do*-leaders), respectively. The leaders for Haverdorf are *direct* leaders.

algorithms to produce non-intersecting leaders in several settings. Fekete and Plaisant [5] 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 [4], in multiple stacks to the left [2], or where the objects to be labeled are polygons rather than points [3]. 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. 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, so 90° angles may rather be avoided.

In this work we study how to label points with labels on one side of the illustration and leaders with at most one bend. Bekos et al. [4] only studied how to minimize the total leader length with rectilinear leaders in this setting; their algorithm runs in $O(n^2)$ time. In this paper we consider other optimization criteria, 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 $O(n \log n)$. We will now state our problem more precisely.

Problem statement. We are given a set P of n points and n disjoint rectangles, possibly of different sizes, called *labels*. The right edges of the labels all lie on a common vertical line, which lies to the left of all points in P. No two labels touch each other.

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 some constant α. If $\alpha = 90°$ the leaders are called *po-leaders*; if $\alpha > 90°$, we call them *do-leaders*[3]. 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

[3] Following the naming scheme of Bekos et al. [4].

of leaders with the underlying map into account. A *labeling* L is a set of n leaders that connects all points to a unique label and all labels 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 $\sum_{l \in L} bad(l)$ is minimized.

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. [4]).

In Section 3 we present algorithms for *do*-leaders ($\alpha > 90°$): 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 present the results of some preliminary experiments with our algorithms, and in Section 5 we briefly discuss possible extensions.

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 90° bends. In Section 2.1 we describe a general 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 no two points lie on a horizontal or a vertical line and no point lies on a horizontal line with an edge of a label (otherwise care should be taken to break ties in a consistent manner).

2.1 A dynamic program 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 an optimal 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 2.

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.

Hence 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, and for each

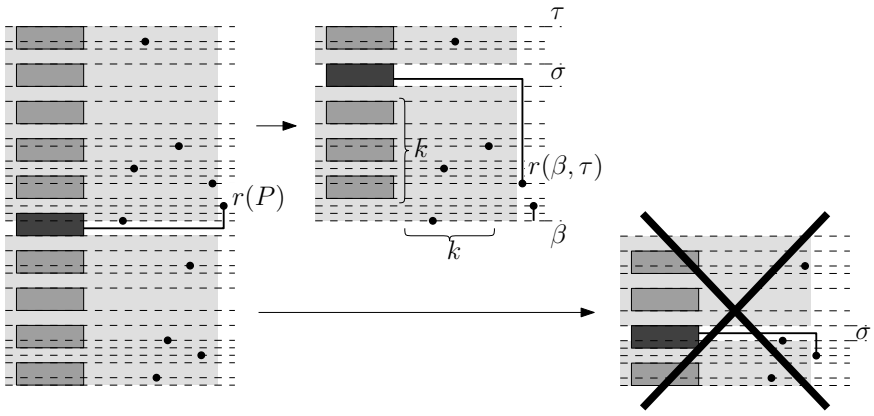


Fig. 2: The recursive structure of an optimal solution. By the choice for the strip that contains the arm of the leader to the rightmost point, the problem is separated into two subproblems. As illustrated by strip σ in the lower subproblem, not all choices for the separating strip σ yield feasible subproblems: in this case there are two points and only one label below σ.

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 2, 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:

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

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 points P, a set of labels 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 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 are solved in $O(n)$ time each, taking $O(n^3)$ time in total. □

2.2 A sweep-line algorithm for minimizing the total leader length

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.*

The idea for proving this lemma is to show that we can eliminate all crossings in L^* by iteratively swapping the labels of two points whose leaders intersect. Any of these swaps does not increase the total leader length; the complete proof can be found in a full version of this paper.

We now 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.

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);

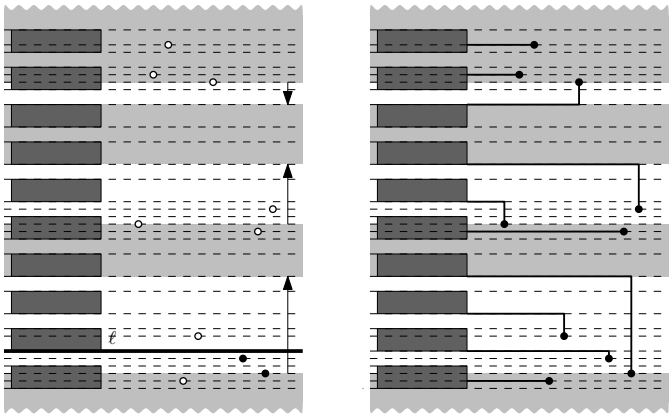


Fig. 3: Left: Classification of strips in the plane sweep algorithm: neutral strips are shaded, downward and upward strips are marked by arrows. When the sweep line reaches the label ℓ, the two black points are in W. Right: The completed minimum-length labeling.

– ℓb_σ: number of labels below σ (incl. any label intersecting σ).

Note that for every strip, $pa_\sigma + pb_\sigma = n$, and $\ell a_\sigma + \ell b_\sigma$ is either n or $n+1$. We classify the strips in three categories and then divide the input into maximal sets of consecutive strips of the same category (see Figure 3):

– *downward*: strips s such that $pa_\sigma > \ell a_\sigma$ (and therefore $pb_\sigma < \ell b_\sigma$);
– *upward*: strips s such that $pb_\sigma > \ell b_\sigma$ (and therefore $pa_\sigma < \ell a_\sigma$);
– *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. Using the leftmost point for labeling 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.

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

The proof of Theorem 2 will be available in a full version of the paper and shows that the algorithm sketched above produces a crossing free labeling of minimum length.

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

In this section we study how to compute an optimal labeling with leaders that have bends with a fixed angle $\alpha > 90°$. In section 3.1 we describe a general solution that works for any badness function $bad()$. In section 3.2 we will give a faster solution 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 makes an angle of 0°, 90°, or α with the x-axis, and no point lies on a horizontal line with an edge of a label (otherwise care should be taken to break ties in a consistent manner).

3.1 A dynamic program 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 rightmost point connected to those leaders. In fact a subproblem was fully defined by specifying the strips β and τ that contain the arms of the leaders: this determined which labels lie inside R, and consequently which point defines the vertical boundary line on the right.

In addition to specify 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 is illustrated by Figures 4a and 4b: 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. The total number of different subproblems may thus increase to $O(n^4)$.

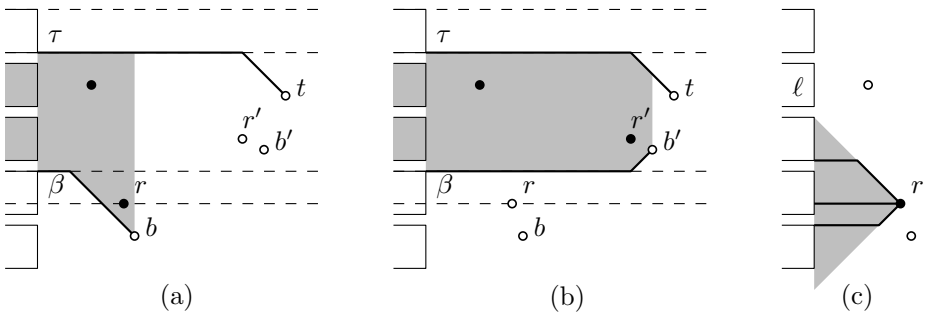


Fig. 4: (a) The subproblem defined by β, τ, b and t. (b) The subproblem defined by β, τ, b' and t. (c) Because leaders have limited slope, no leader from r can reach ℓ.

An additional complication is that as a result of the limited slope of leaders, not every subproblem with the right number of labels and points can be solved—see Figure 4c. The details are easily filled in and we get:

Theorem 3. *Assume we are given a set of points P, a set of labels as described in Section 1, a bend angle α, 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). We can now compute a crossing-free labeling with do-leaders with bend angle α and minimum total badness for P, if such a labeling exists, in $O(n^5)$ time and $O(n^4)$ space.*

3.2 Minimizing the total leader length

Like with *po*-leaders, we can 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 make an angle of α with the x-axis) and *upward diagonals* (lines of positive slope that make an angle of α with the x-axis). For each label ℓ we can define three regions in the plane:

- $A(\ell)$ 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 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 would be (first) hit by a downward diagonal sweep 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 5a.

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 make a leader from ℓ to the lowest point p in $P \cap R(\ell)$ (see Figure 5b and 5c); 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 as described above. We then run the recursive sweep-and-divide algorithm described above. 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 intersections with the algorithm described in the proof of Lemma 1.

Theorem 4. *Assume we are given a set of points P, a set of labels as described in Section 1, and a bend angle α. If there is a labeling for P with do-leaders with*

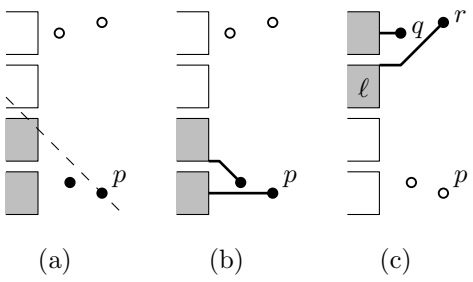


Fig. 5: Illustration of the length-minimization algorithm for *do*-leaders. (a) When the sweep line hits p, we make recursive calls on the input under the sweep line and the input above the sweep line. (b) The result of the recursive call under the sweep line. (c) The result of the recursive call above the sweep line. Although q is the lowest point, ℓ is attached to r, since q cannot reach ℓ.

bend angle α, we can compute a crossing-free 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.

The proof of the correctness of our algorithm is based on the idea to show that any (not necessarily crossing-free) labeling can be transformed into the labeling constructed by our recursive algorithm without increasing the total leader length. Then Lemma 1 can be applied to eliminate the crossings of our solution. The proof will be available in a full version of the paper.

4 Experimental evaluation

We implemented three variants of our algorithms: 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 which is motivated by the observation that a long hand on a short arm looks worse than on a long arm. The parameter λ_{bend} is used to adjust the 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 points with too little clearance. It is defined as

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

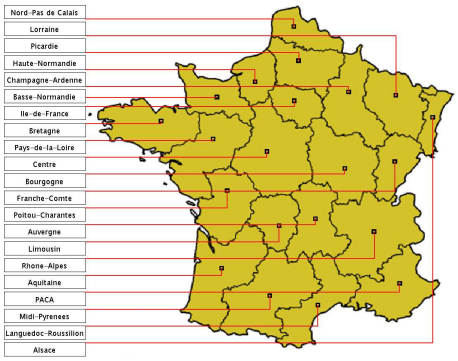
where λ_{cls} is a weight parameter and $d(p,l)$ is the distance between p and l. Adding bad_{cls} helps to reduce confusion when understanding the assignment of points and labels, see Figure 6a generated using bad_{len} and Figure 6b generated using $bad_{\text{len}} + bad_{\text{cls}}$.

We implemented our algorithms as a Java applet[4] and tested them on a map showing the 21 mainland regions of France, see Figure 6. The labelings were computed on an AMD Sempron 2200+ with 1GB main memory, which took between 1 and 5 ms for the *po*-leaders and 12 ms for the *do*-leaders with bend angle $\alpha = 135°$. Running the dynamic programs in a top-down fashion, for *po*-leaders 39% of $O(n^2)$ table entries were computed, while for the *do*-leaders only 0.21% of $O(n^4)$ entries were computed. We also ran the algorithms on artificially generated instances of 100 points uniformly distributed in a unit square. Here the computation of the *po*-leaders took 234 ms averaged over 30 instances and on average 22% of the table entries were computed. The average running time for the *do*-leaders on the same instances was 3328 ms and on average 0.01% of the table entries were computed.

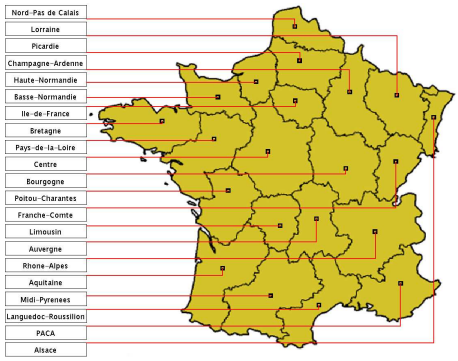
po-leaders vs. do-leaders. Both *po*-leaders and *do*-leaders in Figure 6 have advantages and disadvantages. Obviously, it is not possible to judge whether *po*-leaders or *do*-leaders are generally superior based on our single example map. The answer depends both on the labeled image and on personal 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.

Optimizing for length vs. bends. 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 6b and 6c. In addition, the longer the hand segments are, the harder they are to follow and this is not considered in bad_{bend}. Nevertheless, although direct leaders are easy to read, their number should not be maximized without considering the shape and length of the non-direct leaders. Therefore the hybrid badness function applied in Figures 6d and 6f is designed to find a good compromise between both optimization goals.

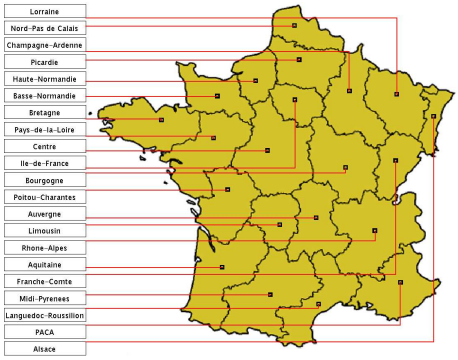
[4] The applet is available at http://i11www.iti.uni-karlsruhe.de/labeling.



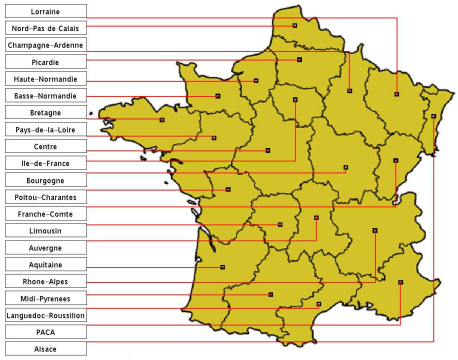
(a) *po*-leaders and badness bad_{len}.



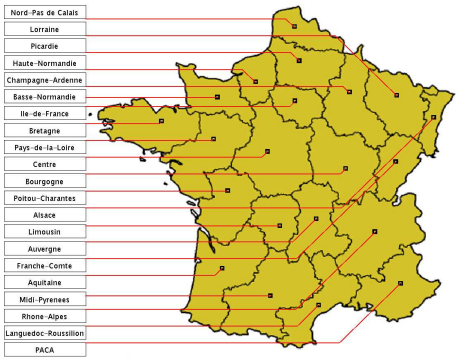
(b) *po*-leaders and badness $bad_{\text{len}} + bad_{\text{cls}}$.



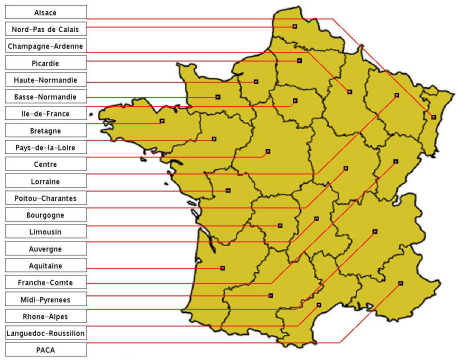
(c) *po*-leaders and badness $bad_{\text{bend}} + bad_{\text{cls}}$.



(d) *po*-leaders and badness $bad_{\text{hyb}} + bad_{\text{cls}}$.



(e) *do*-leaders and badness $bad_{\text{len}} + bad_{\text{cls}}$.



(f) *do*-leaders and badness $bad_{\text{hyb}} + bad_{\text{cls}}$.

Fig. 6: One-sided labelings for the mainland regions of France.

Conclusion. 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. Furthermore the closeness term bad_{cls} turned out to be of great importance for good labelings.

5 Concluding remarks

An interesting future task is to reflect the interference of a leader and the background image in the badness function.

We also looked at the case where the labels are placed on two opposite sides of the point-containing rectangle. Using dynamic programming and similar ideas as for the one-sided case (a split line that splits a subproblem into two two-sided subproblems), we could establish an $O(n^8)$- and $O(n^{14})$-time algorithm for the *po*- and *do*-leaders, respectively. Unfortunately, not only the asymptotical running times of these algorithms were bad, it also turned out that these algorithms are useless in practice since they do not compute a result in acceptable time.

Hence, for producing two-sided labelings in practice we suggest to use the $O(n^2)$-time *po*-leader length-minimization algorithm of Bekos et al. [4] or to split the instance in the middle and solve the resulting 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 an appropriate fashion to find good two-sided *po*- and *do*-labelings. Note that splitting in the middle does in general not yield aesthetically good results. For the *do*-leaders a feasible instance can even become infeasible by splitting in the middle.

References

1. K. Ali, K. Hartmann, and T. Strothotte. Label layout for interactive 3D illustrations. *J. of WSCG*, 13:1–8, 2005.
2. 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 (FSTTCS2006)*, volume 4337 of *Lecture Notes in Computer Science*, pages 81–92, 2006.
3. 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 (APVIS2006)*, volume 60 of *CRPIT*, pages 15–21, 2006.
4. M. A. Bekos, M. Kaufmann, A. Symvonis, and A. Wolff. Boundary labeling: Models and efficient algorithms for rectangular maps. *Computational Geometry: Theory & Applications*, 36:215–236, 2007.
5. 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 (CHI99)*, pages 512–519, 1999.
6. A. Wolff and T. Strijk. The map-labeling bibliography. http://i11www.iti.uni-karlsruhe.de/∼awolff/map-labeling/bibliography/, 2006.