CHAPTER ELEVEN

A Simple and Efficient Algorithm for High-Quality Line Labeling

Alexander Wolff[1] Lars Knipping[2] Marc van Kreveld[3]
Tycho Strijk[3] Pankaj K. Agarwal[4]

11.1 INTRODUCTION

The interest in algorithms that automatically place labels on maps, graphs, or diagrams has increased with the advance in type-setting technology and the amount of information to be visualized. However, though manually labeling a map is estimated to take fifty percent of total map production time (Morrison, 1980), most geographic information systems (GIS) offer only very basic label-placement features. In practice, a GIS user is still forced to invest several hours in order to eliminate manually all label-label and label-feature intersections on a map.

In this chapter, we suggest an algorithm that labels one of the three classes of map objects, namely polygonal chains, such as rivers or streets. Our method is simple and efficient. At the same time, it produces results of high aesthetical quality. It is the first that fulfills both of the following two requirements: it allows curved labels and runs in $O(n^2)$ time, where n is the number of points of the polyline.

In order to formalize what good line labeling means, we studied Imhof's rules for positioning names on maps (Imhof, 1975). His well-established catalogue of label placement rules also provides a set of guidelines that refers to labeling linear objects. (For a general evaluation of quality for label-placement methods, see (van Dijk et al., 1999).) Imhof's rules can be put into two categories, namely *hard* and *soft* constraints. Hard constraints represent minimum requirements for decent labeling:

(H1) A label must be placed at least at some distance ϵ from the polyline.

(H2) The curvature of the curve along which the label is placed is bounded from above by the curvature of a circle with radius r.

[1]Institut für Mathematik und Informatik, Ernst-Moritz-Arndt-Universität, Jahnstraße 15a, D-17487 Greifswald, Germany, `awolff@mail.uni-greifswald.de`

[2]Sender Freies Berlin, Forckenbeckstraße 52, D-14199 Berlin, Germany, `knipping@altus.de`

[3]Department of Computer Science, Utrecht University, The Netherlands, {`marc`, `tycho`}`@cs.uu.nl`, supported by the Dutch Organization for Scientific Research (N.W.O.) and the ESPRIT IV LTR Project No. 21957 (CGAL)

[4]Center for Geometric Computing, Department of Computer Science, Duke University, Durham, NC, U.S.A., `pankaj@cs.duke.edu`, supported by Army Research Office MURI grant DAAH04-96-1-0013, by a Sloan fellowship, by NSF grants EIA–9870724 and CCR–9732787, and by a grant from the U.S.-Israeli Binational Science Foundation.

(H3) The label must neither intersect itself nor the polyline.

Soft constraints on the other hand help to express preferences between acceptable label positions. They formalize aesthetic criteria and help to improve the visual association between line and label. A label should

(S1) be close to the polyline,

(S2) have few inflection points,

(S3) be placed as straight as possible, and

(S4) be placed as horizontally as possible.

We propose an algorithm that produces a candidate strip along the input polyline. This strip has the same height as the given label, consists of rectangular and annular segments, and fulfills the hard constraints. In order to optimize soft constraints, we use one or a combination of several evaluation functions.

The candidate strip can be regarded as a simplification of the input polyline. The algorithm for computing the strip is similar to the Douglas-Peucker line-simplification algorithm (Douglas and Peucker, 1973) in that it refines the initial solution recursively. However, in contrast to a simplified line, the strip is never allowed to intersect the given polyline. The strip-generating algorithm has a runtime of $O(n^2)$, where n is the number of points on the polyline. The algorithm requires linear storage.

Given a strip and the length of a label, we propose three evaluation functions for selecting good label candidates within the strip. These functions optimize the first three soft constraints. Their implementation is described in detail in (Knipping, 1998). We can compute in linear time a placement of the label within the strip so that the *curvature* or the *number of inflections* of the label is minimized. Since it is desirable to keep the label as close to the polyline as possible (while keeping a minimum distance) we also investigated the *directed label-polyline Hausdorff distance*. This distance is given by the distance of two points; a) the point p on the label that is furthest away from the polyline and b) the point p' on the polyline that is closest to p. Under certain conditions we can find a label position that minimizes this distance in $O(n \log n)$ time (Knipping, 1998). Here we give a simple algorithm that finds a near-optimal label placement according to this criterion in $O(nk + k \log k)$ time, where k is the ratio of the length of the strip and the maximum allowed discrepancy to the exact minimum Hausdorff distance.

If a whole map is to be labeled, we can also generate a set of near-optimal label candidates for each polyline, and use them as input to general map-labeling algorithms as (Edmondson et al., 1997; Kakoulis and Tollis, 1998; Wagner and Wolff, 1998). Some of these algorithms accept a priority for each candidate; in our case we could use the result of the evaluation function.

In his list of guidelines for good line labeling, Imhof also recommends the labeling of a polyline at regular intervals, especially between junctions with other polylines of the same width and color. River names e.g. tend to change below the mouths of large tributaries. This problem can be handled by extending our algorithms as follows. We compute our strip and generate a set of the, say ten best label candidates for each river segment that is limited by tributaries of equal type. Then we can view each river segment as a separate feature, and again use a general map-labeling algorithm to label as many segments as possible. Prioritizing each label candidate with its distance to the closer end of the river segment would give candidates in the middle of a segment a higher priority and thus tend to increase label-label distances along the polyline.

This chapter is structured as follows. In the next section we briefly review previous work on line labeling. In Section 11.3 we explain how to compute a buffer around the input polyline that protects the strip from getting too close to the polyline and from sharp bends at convex vertices. In Section 11.4 we give the algorithm that computes the strip and in Section 11.5 we show how this strip can be used to find good label candidates for the polyline. Finally, in Section 11.6 we describe our experiments. Our implementation of the strip generator for x-monotonous polylines and the three evaluation functions can be tested on-line at the URL *http://www.inf.fu-berlin.de/map-labeling/lines*.

11.2 PREVIOUS WORK

For an extensive bibliography about map labeling in general, see (Wolff and Strijk, 1996). The problem of automated line labeling has been treated before. In (Doerschler and Freeman, 1992; Barrault and Lecordix, 1995; Alexander and Hantman, 1995; Edmondson et al., 1997; Kramer, 1997) only rectangular labels are allowed; curved labels are not considered. In (Freeman, 1988) a set of label-placement rules similar to those of (Imhof, 1975) is listed, followed by a rough description of an algorithm. An analysis of Figure 8 in (Freeman, 1988) shows that river names are broken into shorter pieces that are then placed parallel to segments of the river. Each piece ends before it would run into the river or end too far from the current river segment.

In (Barrault, 1997) curved labels are taken into account. First, an input polyline is split into sections depending on its length and junctions (forks) with other polylines. For details of this step, see (Barrault and Lecordix, 1995). Then the polyline is treated with an adaptation of an operator from morphological mathematics, closure, that is a mixture of an erosion and a dilation. This operator yields a baseline for label candidates where the polyline does not bend too abruptly. It is not clear how this is done algorithmically; no asymptotic runtime bounds are given. Finally, simulated annealing is used in order to find a good global label placement, i.e. a placement that maximizes the number of features that receive a label and at the same time takes into account the cartographic quality of each label position.

In (Poon et al., 1998) a more theoretical problem is analyzed; an instance of axis-parallel line segments is labeled with rectangular labels of common height. While the length of each label equals that of the corresponding line segment, the label height is to be maximized.

While the restriction to rectangular labels is acceptable for technical maps or road maps (where roads must be labeled with road numbers), we feel that curved labels are a necessity for high-quality line labeling. The method we suggest is the first that fulfills both of the following two requirements: it allows curved labels and its runtime is in $O(n^2)$. The runtime thus only depends on the number of points of the polyline, and not on other parameters such as the resolution of the output device. Note that the time bound holds even if the approximate Hausdorff distance is used to select good label candidates within the strip as long as we choose the parameter k linear in n.

11.3 A BUFFER AROUND THE INPUT POLYLINE

In order to reduce the search space for good label candidates, we generate a strip along the input polyline that is (a) likely to contain good label positions and (b) easy to compute. Generating our strip consists of two major tasks. First, we compute a buffer around the polyline that our strip must not intersect. Second, we generate an initial strip and refine it recursively. Each refinement step brings the strip closer to the polyline, but also introduces additional inflections.

The input to our algorithm consists of a polyline $P = (p_1, \ldots, p_n)$ with points $p_i = (x_i, y_i)$, a minimum label-polyline distance ε, a maximum curvature $1/r$, and a label height h. It makes sense to choose $r \gg \varepsilon$ but the algorithm does not depend on this. We assume that P is x-monotonous, i.e. $x_1 < \ldots < x_n$. Non-monotonous polylines can be split up into monotonous pieces of maximum length in linear time by a simple greedy algorithm. That algorithm goes sequentially through the edges of the polyline. Whenever adding the current edge to the current piece would make that piece non-monotonous, a new piece is started with the current edge.

For ease of presentation we direct P from p_1 to p_n and only label the upper (i.e. left) side of the polyline. We use r-disk (r-arc) as shorthand for a disk (arc) of radius r. We say that p_i is *at a right turn of* P if p_{i+1} lies to the right of the directed line through p_{i-1} and p_i, see p_3 or p_4 in Figure 11.1.

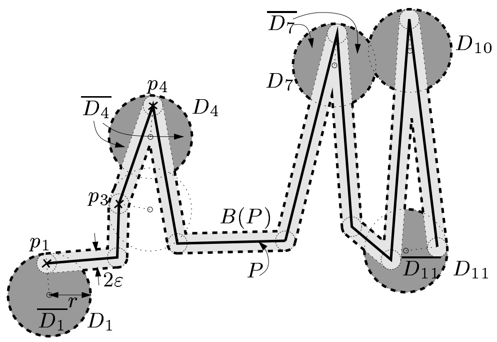

Figure 11.1 The boundary of the (ε, r)-buffer $B(P)$ (bold dashed line) of the input polyline P (bold solid line).

We define the *(ε, r)-buffer* $B(P)$ in two steps. First let the *ε-buffer* be the union of all ε-disks whose center lies on P, see the light-shaded area in Figure 11.1. Second we add certain pieces of r-disks D_i placed at right turns p_i of P . Their task is to bound the curvature of our strip. The center m_i of D_i is placed on the angular bisector b_i of the adjacent edges of P such that D_i touches and contains the ε-disk centered at p_i, see Figure 11.2. Let $\overline{D_i}$ be the part of D_i that is left of the ε-buffer and touches the ε-disk, see the dark-shaded areas in Figure 11.1. Then $B(P)$ is the union of the ε-buffer and the $\overline{D_i}$ for each right turn p_i.

To simplify the calculation of the strip, we also place r-disks D_1 and D_n at the endpoints p_1 and p_n of P, respectively. Let b_n be the normal to the edge $p_{n-1}p_n$ in p_n. Then the center of D_n lies on b_n such that D_n touches and contains the ε-disk centered at p_n, see D_n in Figure 11.2. The placement of D_1 is analogous.

In order to compute the boundary of the (ε, r)-buffer we first compute that of the ε-buffer. This is simple since the x-monotonicity of P guarantees that the ε-buffer does not have any holes.

For computing the candidate strip it is important that we have access to the elements of the outer face of the (ε, r)-buffer in the order in which they occur. We compute the (ε, r)-buffer in two phases.

In the first phase, for each right turn p_i we follow the boundary of the ε-buffer from t_i to the right until we intersect the boundary of D_i for the first time. This intersection point is denoted by r_i, see Figure 11.2. The arc from t_i to r_i, oriented clockwise, is the *right arc* R_i, one of the two parts of the boundary of D_i we are interested in. The *left arcs* L_i that go counterclockwise from t_i to l_i can be computed analogously. A special case arises if t_i lies in the interior of the ε-buffer. Then R_i or L_i is empty, and we have to follow P from p_i in both directions until we arrive at a point or edge that corresponds to an arc or line segment on the upper part of the ε-buffer. From there, we can continue as usual.

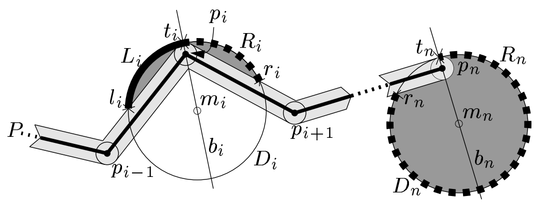

Figure 11.2 Placing r-disks D_i at right turns p_i of the input polyline P.

Clearly, this procedure has a worst-case runtime of $O(n^2)$. The worst case occurs if there are a linear number of right turns p_i where we have to walk over a linear number of segments of the ε-buffer until we hit l_i or r_i, i.e. if r is large compared to the length of the edges of P. However, in practice one can expect to walk only over a constant number of segments of the ε-buffer; then the running time is $\Theta(n)$, see Section 11.6. The worst-case running time can be improved using more sophisticated data structures, but we omit this improvement here as it makes the algorithm more complicated.

In the second phase, we incrementally extend the ε-buffer to the (ε, r)-buffer using the left and right arcs we just computed. We maintain $\mathcal{B}_{\text{curr}}$, the outer face of the union of the ε-buffer and the areas $\overline{D_i}$ we have processed so far. Initially let $\mathcal{B}_{\text{curr}}$ be the boundary of the ε-buffer and let the interior of $\mathcal{B}_{\text{curr}}$ the interior of the ε-buffer. Let the r-arc A_i be the union of L_i and R_i. Note that A_i is the part of the boundary of $\overline{D_i}$ that is a potential part of the outer face of the (ε, r)-buffer. For each right turn p_i we check whether A_i lies completely in the interior of $\mathcal{B}_{\text{curr}}$. If this is not the case we extend $\mathcal{B}_{\mathrm{curr}}$ by using the appropriate parts of A_i.

The boundary of the ε-buffer consists of a linear number of line and arc segments to which we add $O(n)$ arcs of type A_i. One can prove that each of these arcs can contribute at most three pieces to the outer face of $B(P)$. Our implementation does not depend on this result, but it shows that the outer face of $B(P)$ has linear complexity. Due to the incremental construction this is also an upper bound for the size of $\mathcal{B}_{\mathrm{curr}}$.

Given these observations it is easy to devise an $O(n^2)$-algorithm that computes the boundary of the outer face of $B(P)$. We store $\mathcal{B}_{\mathrm{curr}}$ in a doubly connected list. Since the length of this list is linear we can afford to scan the whole list when we search for intersections with the current arc A_i. If we consider carefully whether we enter or leave the interior of the area delimited by $\mathcal{B}_{\mathrm{curr}}$, we can update $\mathcal{B}_{\mathrm{curr}}$ in linear time for each right turn. We omit details here.

In our implementation of the second phase we use a similar trick as in the first phase to avoid a quadratic runtime in many cases. We exploit the fact that an arc A_i usually spans only a constant number of elements of $\mathcal{B}_{\mathrm{curr}}$.

11.4 A CANDIDATE STRIP

Once we have the outer face of the (ε, r)-buffer, we compute the baseline of the label candidate strip and refine it recursively. We refer to the line and arc segments that delimit the buffer on the upper side between l_1 and r_n as baseline *objects*. We have access to these objects in the order in which they appear on the boundary of the buffer's outer face. We start with an arc A that touches the first and last object O_i and O_k, respectively. We bend A towards the buffer until it hits a third object O_j. There, we split A into two pieces, its children. We connect the children of A with a piece of O_j that initially has length zero. Then we recursively bend the children further towards the buffer, see Figure 11.3. While we bend, the portion of O_j that connects the children of A is growing. Note that there are two phases: in the first, the radius of the arcs increases while it decreases in the second. The recursion ends where O_i and O_k are adjacent on the buffer (since there is no O_j then) and in the second phase where the curvature of an arc would exceed $1/(r+h)$, h the label height.

For each level of the recursion, the sequence of arcs we obtain in this way forms a continuous curve L. If we direct L from left to right, it becomes obvious that the radius of all arcs that turn right (i.e. towards the buffer) is at least r and the radius of arcs that turn left is at least $r+h$. By using L as the baseline of our strip of height h we ensure that all arcs that form the upper boundary and the baseline of the strip have at least radius r. Thus the strip fulfills the curvature constraint H2. Since the baseline of the strip cannot intersect the ε-buffer it is clear that the strip also fulfills the distance constraint H1. The non-self-intersection constraint H3 can easily be kept by ending the recursion where the distance between O_i and O_k is less than $2h$.

If the number of inflections is to be kept small, the recursion can also be stopped whenever the directed distance of a strip segment to the polyline is below a given threshold. However this is difficult to check without the Voronoi diagram

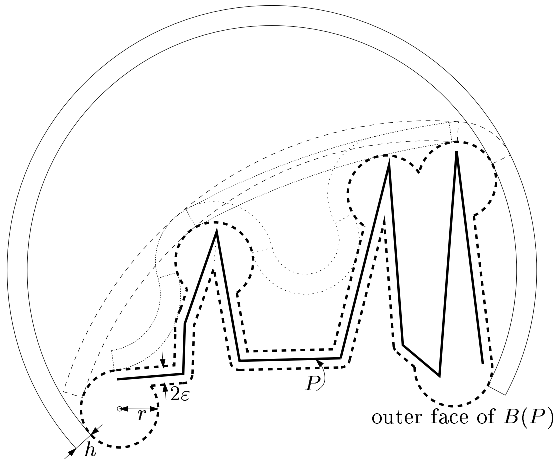

Figure 11.3 refining the candidate strip: first level (solid), second level (dashed), third level (densely dotted), and forth level (dotted)

of the points and (open) edges of P.

It is possible to add two interesting refinement levels. In both, an arc of the baseline does not necessarily touch three objects on the boundary of the buffer's outer face. For a strip with more rectangular segments one could add a refinement level between level 2 and 3 of the leftmost strip segment in Figure 11.3. Note that the radii of the annular strip segments there increase up to level 2 and then decrease again. Rectangular segments in an additional refinement level can thus be viewed as annular segments with infinite radius. On the other hand, to make the strip follow P as closely as possible, a final refinement level could be added where *all* annular strip segments are delimited by two arcs with radius r and $r + h$. The baseline of this strip is part of the curve on which a disk of radius $r + h$ is rolled around the buffer if the disk must always touch the buffer but not intersect its interior.

In order to determine the third object on an arc, we test each object between the left- and rightmost object in constant time. Thus we need linear time for each level of the recursion. As with the Douglas-Peucker line-simplification algorithm, the number of recursion levels depends on the distribution of the input data and can vary from $\Omega(\log n)$ to $O(n)$. Given the outer face of the (ε, r)-buffer the strip can hence be computed in $O(n^2)$ time, while the average case can be expected to be in $O(n \log n)$.

11.5 FINDING GOOD LABEL POSITIONS

In order to satisfy the soft constraints, we evaluate label candidates within the strip according to curvature, number of inflections, or directed label-polyline Hausdorff distance. (We define the curvature of a label as the sum over curvature times length of each label segment. The curvature of a rectangular segment is 0; that of an annular segment with arcs of radius r_1 and $r_2 = r_1 + h$ is $1/r_1$.) For all three evaluation functions, the basic idea is the same. We discretize the space of label candidates such that the discrete space has linear size and contains minima. Then we search the discrete space for a minimum.

For curvature and number of inflections it is easy to see that there is a minimizing label candidate that starts or ends with one of the rectangular or annular segments of the strip. In order to find a minimum, we push a label of the given length through the strip and stop whenever a new segment starts (or ends). To compute the measure of the current candidate, we only have to do a constant number of updates given the value at the previous position. This is how we can find a placement minimizing curvature or number of inflections in linear time.

For Hausdorff distance, the discretization is more difficult. We only take into account the baseline of the strip. In order to compute efficiently the distance between the baseline of a label candidate and the polyline P, we need to know the closest object (point or edge) of P for every point on the whole baseline. Intersecting the baseline with the Voronoi diagram of the objects of P would yield this information and lead to an $O(n \log n)$ algorithm under certain conditions (Knipping, 1998).

However, computing the Voronoi diagram for a set of points and line segments is not a trivial task in practice. Therefore we implemented a simpler algorithm that finds a near-optimal label placement as follows. Given an integer k, we split the baseline into k pieces of equal length. Let γ be the length of such a piece. We approximate the distance between each piece and P by the distance of the piece's midpoint from P. This can be done by brute force in $O(nk)$ time with $O(k)$ storage. Then we proceed as above: we push the label through the strip, stop at each midpoint and evaluate the current label position. Its Hausdorff distance to P is within γ from the maximum over the distances of all baseline pieces covered by the label. For fast access to this approximate maximum, we keep the appropriate distances in a priority queue. During the execution of the algorithm, we must insert the distance of each piece at most once into the queue. The same holds for deletions. Each such operation costs $O(\log k)$ time, hence we can compute an optimal placement among all those starting at a midpoint of a baseline piece in $O(nk + k \log k)$ time with $O(k)$ storage. The triangle inequality guarantees that this placement is at most γ further away from P than a placement minimizing the exact directed Hausdorff distance. A detailed description of the implementation of the above evaluation functions can be found in (Knipping, 1998) (in German).

11.6 EXPERIMENTAL RESULTS

In order to analyze our line-labeling algorithm, we applied it to synthetic and to real-world data. The latter is taken from the CIA-map data at the URL *ftp://gatekeeper.dec.com/pub/graphics/data/cia-wdb/db.tar.Z*, see Figures 11.4 and 11.5. In both figures, labels were placed according to the approximated Hausdorff distance.

The synthetic data belongs to three different example classes. Due to lack of space we can only present our results on one class. For more detailed information including graphs depicting the frequency of crucial operations, see our Web page.

For the example class **RandomWalk** we use random numbers Δx_i and Δy_i

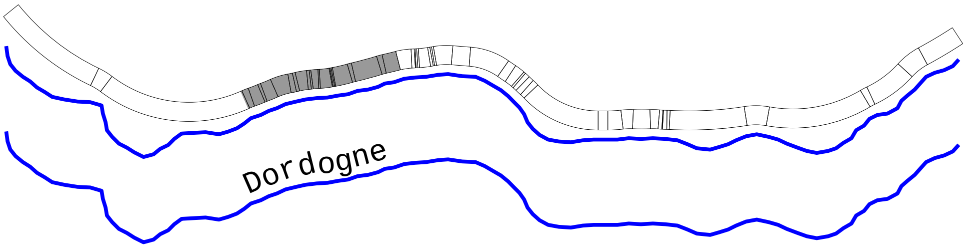

Figure 11.4 A piece of the Dordogne (109 points). Above with candidate strip and label placement (shaded grey), below with lettering

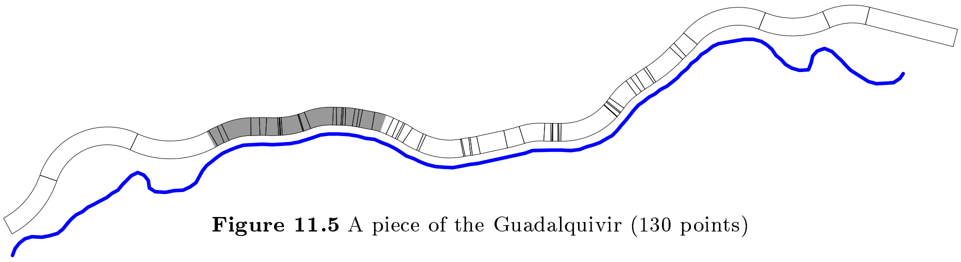

Figure 11.5 A piece of the Guadalquivir (130 points)

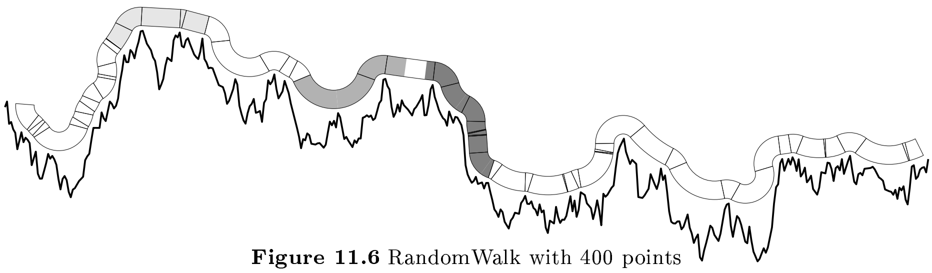

Figure 11.6 RandomWalk with 400 points

that we draw from a normal distribution with mean 0 and standard deviation 1. In order to get an x-monotonous polyline we choose the x-coordinates as follows: $x_1 = 0$ and $x_i = x_{i-1} + |\Delta x_i|$. Then we scale all x_i by x_n such that $0 = x_1 < x_2 < \ldots < x_n = 1$. Similarly, we set the y-coordinates to $y_1 = 0$ and $y_i = y_{i-1} + \Delta y_i/100$.

Figure 11.6 shows an instance of RandomWalk with the candidate strip of the last refinement level, not counting the additional levels mentioned in Section 11.4. The grey regions indicate an optimal label placement within the strip minimizing curvature, number of inflections, and approximative Hausdorff distance (left to right, shaded light to dark). The parameters for the strip computation were minimum label-polyline distance $\varepsilon = 0.005$, curvature bound $r = 0.01$, and label height $h = 0.02$. More examples can be found in (Knipping, 1998) or generated on our Web page.

We generated 50 RandomWalk examples with 100, 200, …, 1000 points to analyze the performance of our C++ implementation. We used the SunPRO-CC

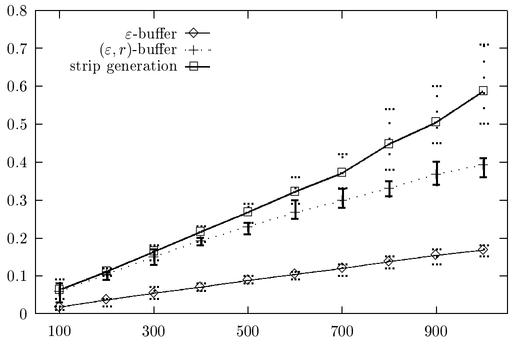

Figure 11.7 Strip generation time

compiler with optimizer flags `-fast -O3` and measured runtimes on a Sun Ultra-Sparc 250. We prepared two graphs, see Figures 11.7 and 11.8. In both, the y-axis gives the average CPU time (in seconds) and the x-axis gives the number n of points of the polyline. The points on our graphs give the results averaged over all 50 examples; the extent of the vertical bars indicates the minimum and maximum runtime among these 50 examples.

Figure 11.7 shows the running times of the ε-buffer, (ε, r)-buffer and strip generation for RandomWalk. Note that the three curves are additive; i.e. the top-most curve corresponds to the total runtime. The two additional refinement levels mentioned in Section 11.4 were included.

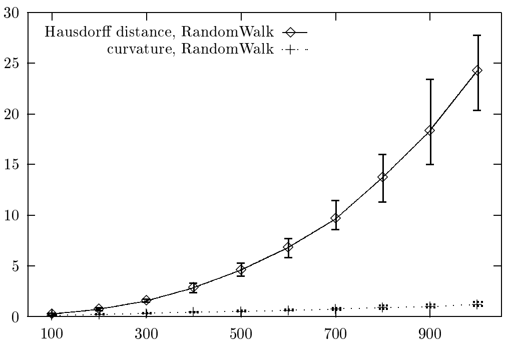

Figure 11.8 Running times for label placement

In Figure 11.8 we give the runtimes for placing labels within the pre-computed strip according to curvature and approximated Hausdorff distance. Here the parameters were curvature $r = 8/n$, minimum distance $\varepsilon = 2/n$, label height $h = 10/n$, and label length $\ell = 50/n$. For minimizing the Hausdorff distance we set the approximation parameter γ to $1/(2n)$. We omitted the curve for number of inflections since it is identical to that of curvature. Other than in the description in Section 11.5 we used lists instead of priority queues for the approximated Hausdorff distance, hence the quadratic runtime behaviour.

11.7 CONCLUSION

We have presented a new and conceptually simple method for high-quality line labeling. It is the first that fulfills both of the following two requirements: it allows curved labels and its worst-case runtime is in $O(n^2)$. We introduced a concept of gradual refinement that is similar to the idea of the Douglas-Peucker line-simplification algorithm. This concept allows to introduce additional application-dependent criteria and to stop the refinement when these criteria are met.

An experimental evaluation of our algorithm shows that it usually runs in sub-quadratic time and generally yields good results in practice. However, since we reduce the search space for good label candidates to a one-dimensional strip, it is clear that we cannot hope to find an optimal label placement in every case. As the following example indicates, a more flexible strategy in the buffer construction might help to overcome problems caused by the reduction of the search space.

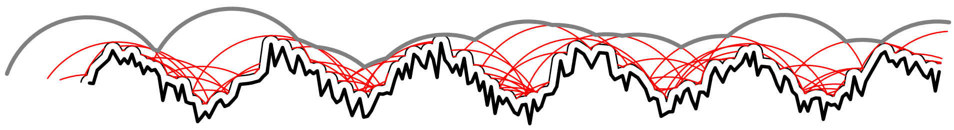

Figure 11.9 Disturbing effects of the definition of the (ε, r)-buffer. (The upper part of its outer face is marked by bold grey arcs; the input polyline below consists of bold black line segments.)

In Figure 11.9 we depicted all r-arcs at right turns of the input polyline P. The parameter r was chosen large compared to the average segment length of P. As a result, some of the arcs that contribute to the (ε, r)-buffer are quite distant from the input polyline P. They were caused by right turns incident to two very steep but short edges of P. It would be desirable to remove these arcs. However, we must ensure that the resulting strip does not violate the curvature constraint H2. This can be done as follows. After the first phase of the (ε, r)-buffer computation we compute the directed Hausdorff distance of each r-arc A_i to the ε-buffer between l_i and r_i. In order of descending distance we check for each A_i whether the corresponding ε-arc lies completely in the area $\overline{D_j}$ of another r-arc A_j. If this is the case, we remove A_i. Then we proceed to the second phase of the buffer computation as usual. Note that the resulting outer face of the buffer still consists exclusively of r-arcs and line segments. Thus the strip will still keep H2.

An alternative approach is as follows. We observed that our placement of the r-disks is good if the the adjacent edges of the polyline are long enough. Then the directed Hausdorff distance between the arc A_i and the ε-buffer is minimized. However, in general the placement of the r-disks is too inflexible. It could certainly be improved if we tried to minimize the aformentioned distance during the placement. Then the placement of the r-disks would take into account not only the adjacent edges of the polyline but all of the polyline (or the ε-buffer) between l_i and r_i.

Finally we would like to acknowledge a simple and elegant idea of Mike Lonergan, University of Glamorgan, Pontypridd. He suggested to put the ε-buffer around the label (and thus simply thicken the strip by 2ε) instead of the polyline. Unfortunately, this does not solve the problem of placing the r-circles.

REFERENCES

Alexander, D. H. and Hantman, C. S., 1995, Automating linear text placement within dense feature networks. In *Proc. Auto-Carto 12*, Charlotte, NC, U.S.A., pages 311–320.

Barrault, M., 1997, An automated system for name placement which complies with cartographic quality criteria: The hydrographic network. In *Proceedings of the Conference on Spatial Information Theory*, volume 1329 of *Lecture Notes Comput. Sci.*, (Springer-Verlag), pages 499–500.

Barrault, M. and Lecordix, F., 1995, An automated system for linear feature name placement which complies with cartographic quality criteria. In *Proc. Auto-Carto 12*, Charlotte, NC, U.S.A., pages 321–330.

Doerschler, J. S. and Freeman, H., 1992, A rule-based system for dense-map name placement. *Communications of the ACM*, 35:68–79.

Douglas, D. H. and Peucker, T. K., 1973, Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Canadian Cartographer*, 10(2):112–122.

Edmondson, S., Christensen, J., Marks, J., and Shieber, S., 1997, A general cartographic labeling algorithm. *Cartographica*, 33(4):13–23.

Freeman, H., 1988, An expert system for the automatic placement of names on a geographic map. *Information Sciences*, 45:367–378.

Imhof, E., 1975, Positioning names on maps. *The American Cartographer*, 2(2):128–144.

Kakoulis, K. G. and Tollis, I. G., 1998, A unified approach to labeling graphical features. In *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, pages 347–356.

Knipping, L., 1998, Beschriftung von Linienzügen. Master's thesis, Fachbereich Mathematik und Informatik, Freie Universität Berlin. Available at *http://www.inf.fu-berlin.de/map-labeling/papers/k-bl-98.ps.gz*

Kramer, J. C., 1997, Line feature label placement for ALPS5.0.

Morrison, J. L., 1980, Computer technology and cartographic change. In Taylor, D., editor, *The Computer in Contemporary Cartography.* J. Hopkins Univ. Press, New York.

Poon, C. K., Zhu, B., and Chin, F., 1998, A polynomial time solution for labeling a rectilinear map. *Information Processing Letters*, 65(4):201–207.

van Dijk, S., van Kreveld, M., Strijk, T., and Wolff, A., 1999, Towards an evaluation of quality for label placement methods. In *Proceedings of the 19th International Cartographic Conference*, Ottawa, Int. Cartographic Association, pages 905–913.

Wagner, F. and Wolff, A., 1998, A combinatorial framework for map labeling. In Whitesides, S. H., editor, *Proceedings of the Symposium on Graph Drawing '98*, volume 1547 of *Lecture Notes in Computer Science*, (Springer-Verlag), pages 616–331.

Wolff, A. and Strijk, T., 1996, A map labeling bibliography. *http://www.inf.fu-berlin.de/map-labeling/bibliography/.*