Dynamic One-Sided Boundary Labeling

Martin Nöllenburg
Karlsruhe Institute of Technology and
University of California, Irvine
noellenburg@kit.edu

Valentin Polishchuk Mikko Sysikaski
Helsinki Institute for Information Technology
CS Department, University of Helsinki
firstname.lastname@cs.helsinki.fi

ABSTRACT

In boundary labeling, features on a map are connected to a stack of labels on the map boundary, using simple polylines called *leaders*. We consider the setting that the labels are axis-aligned non-overlapping rectangles placed on one side of the map, and leaders are rectilinear polylines with at most one bend. The goal is to find a labeling that minimizes the total length of the leaders.

We introduce three extensions of the one-sided boundary labeling problem: (i) a *dynamic* setting for continuous scale changes, (ii) a *clustered* setting for multiple label stacks, and (iii) a combined *dynamic clustered* setting. We obtain the following results:

- Optimal label placement as a function of map scale can be computed in $O(n \log n + \sigma \log n)$ time, where σ is the number of "combinatorially different" labelings that occur during zooming.
- In a map with fixed scale, an optimal clustered label placement can be found in $O(n \log n)$ time.
- In $O(n \log^2 n + \gamma \log n)$ time one can build a structure of size $O(\gamma)$ representing the optimal clustered label placement for *all* possible map scales; here γ is, again, the number of combinatorially different labelings.

We further extend our basic model to the case where labeled features enter or leave the viewport due to map panning and zooming. Our algorithms are based on combining standard computational-geometry tools and have been implemented in a Java applet (available online), which indicates that the algorithms are fast enough for interactive use without delays.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*

General Terms

Algorithms, Experimentation

GIS '10, 03-NOV-2010, San Jose CA, USA

Keywords

Map labeling, continuous viewport changes

1. INTRODUCTION

Many different kinds of visualizations—medical illustrations, engineering diagrams, statistical graphics, maps—annotate features of interest by textual labels. These labels are usually required to be placed directly next to the features without overlapping each other or occluding important parts of the illustration. If, however, too many features appear densely together or the labels are too large, it may not be possible to place the labels in this fashion. A common alternative approach is thus to place the labels next to one or more sides of the figure's boundary and to connect them to the respective features using low-complexity curves. Labels in this case are also commonly referred to as *call-outs* and the curves connecting call-outs and features are called *leaders*. For a static graphic, the algorithmic problem of placing the labels and leaders so that a certain quality measure is optimized is called the *static boundary labeling problem*. A natural objective for creating labelings of low visual complexity is to minimize the total leader length and to avoid intersecting leaders. Figure 1 shows an example.

Of increasing importance in geovisualization are *dynamic* maps, in which the user can interactively select regions of interest by zooming and panning his individual view of the illustration. Another use of dynamic visualizations are location-aware maps on mobile devices, which continuously change with movement. In this paper we address the problem of computing and maintaining an optimal placement of labels and leaders in a map whose scale changes continuously; we call this the *dynamic boundary labeling problem*. We also consider changes of the view—in particular, changes to the set of visible features—that occur due to map panning.

Related Work.

Over the last decades, most research efforts were devoted to labeling point features in static maps by directly placing non-overlapping labels next to the features. This problem is known to be NP-hard and many heuristics and approximation algorithms exist, see [19] and the extensive bibliography on map labeling maintained by Wolff and Strijk [20]. The (static) boundary labeling problem was first introduced as an algorithmic problem by Bekos et al. [5] and subsequently studied in different settings for rectilinear and diagonal leader shapes with one or two bends and label positions on one, two, or four sides of the map [3,6,14]; placing the labels in multiple columns on one side of the map was also considered [4]. Still,

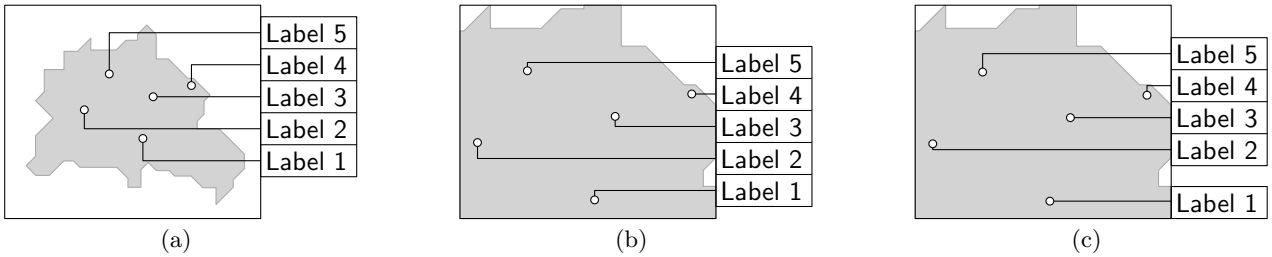

Figure 1: A sample instance with five points shown at a small scale (a) and at a larger scale (b,c). Labels are in a single stack (a,b) or clustered (c).

all previous results assume that labels should be as large as possible. Thus, they form a single stack of labels occupying the full height (or width) of all available boundaries.

In recent years, *dynamic* map labeling using internal labels has been studied. Petzold et al. [15, 16] use a preprocessing step to generate a reactive conflict graph that represents possible label overlaps for all scales. For any fixed scale and map region, a conflict-free labeling can be computed quickly using heuristic methods. Continuous movement and zooming, however, are not explicitly supported by their methods and may lead to sudden discrete changes of label positions. Been et al. [1, 2] define consistency criteria for dynamic labelings that avoid popping and jumping of labels during movement and zooming. They show NP-hardness of maximizing the number of labels in a consistent labeling and present several approximation algorithms for the problem.

Our Contributions.

We combine the two concepts of boundary labeling and dynamic map labeling into *dynamic boundary labeling*. More precisely, we want to minimize the total leader length in one-sided boundary labelings (all labels are on the same side of the map) with one-bend rectilinear leaders. We first study a dynamic version of *stacked* boundary labelings, where all labels occupy a contiguous interval of the boundary (but not necessarily the whole boundary). The problem is to find the optimum position of the label stack as a function of the map scale. Figures 1a and 1b show two different positions for different scales. In Section 3 we show that for this problem the optimal label placement can be computed in $O(n \log n + \sigma \log n)$ time where σ is the complexity of the problem instance in terms of the number of combinatorially different labelings.

Our second problem introduces *clustered* instead of stacked boundary labelings. Here we allow irregular gaps between consecutive clusters of labels in order to decrease the total leader length. Figure 1c shows an example, where the optimal label placement uses two clusters. In Section 4 we give a simple $O(n \log n)$-time algorithm that determines the optimal number of label clusters and their placement for the static boundary labeling problem.

Finally, in Section 5, we present our main result and combine the dynamic setting and the clustered setting. We compute, in $O(n \log^2 n + \gamma \log n)$ time, a structure that represents the optimal clustered label placement as it changes during zooming; here γ is again a measure of the complexity of the instance.

It is worth noting that our algorithms are *output-sensitive*: $\Omega(\sigma)$ space is needed to represent the changes in a stacked labeling, and $\Omega(\gamma)$ space is needed for the changes in a clustered labeling. Our algorithms spend only a logarithmic time per change to update the optimal labeling.

In Section 6 we address two important practical issues that are caused by the limited size of a map displayed on screen. With these issues taken into account, we implemented our algorithms and made them available (including the source code) as a Java applet located at `http://cs.helsinki.fi/group/compgeom/boundarylabeling/`. Experimental results for randomly distributed sites indicate that the output complexities σ and γ grow almost linearly with n.

2. PRELIMINARIES

In this section we extend the existing model for boundary labeling to models for dynamic, for clustered, and for dynamic clustered boundary labeling. Additionally, we repeat two line-sweeping algorithms on which our methods are based.

2.1 Model

Let $P = \{p_1, \ldots, p_n\}$ be a set of points in the plane, called *sites*, that are to be labeled. Suppose that the user's screen has width w_s and height h_s and thus *aspect ratio* $r = w_s/h_s$. Any rectangle R that has aspect ratio r and that contains the sites P is called a *view* of P. In a mapping application that displays R on the screen we need to scale R by some factor $1/z > 0$ in order to match the screen size. The larger the rectangle R the smaller the scale $1/z$, which corresponds to the cartographic definition of scale. We define the inverse z of the scale $1/z$ to be the *zoom level* of the view.

Let $\mathbb{L}$ be the vertical line containing the right side of the rectangle R. A *right-sided boundary labeling* of R is a set of n axis-aligned non-overlapping rectangular labels $l_1, \ldots, l_n$ whose left sides lie on $\mathbb{L}$. We make the simplifying assumption that all labels have uniform height; our algorithms can easily be adapted to deal with non-uniform labels. Typically [14], all n labels form a stack, i.e., there are no or only fixed-height gaps between any two adjacent labels. Each site p is connected to the fixed anchor point of a distinct label l on $\mathbb{L}$ by a simple curve called *leader*. In this paper, we assume that a label's anchor point is the vertical midpoint of the label's left side. As leaders we consider one-bend rectilinear polylines that start with an optional vertical segment and then extend horizontally to the anchor point of a label. Such leaders are called *po*-leaders in the literature [5] as they first run ***p***arallel to and then ***o***rthogonal to $\mathbb{L}$. We call a leader with a vertical segment of length 0 a *direct* leader. Otherwise, a leader whose bend is above (below) its site is

called *upward* (*downward*). For example, in Fig. 1b leader 4 is direct, leaders 1 and 5 are upward and leaders 3 and 2 are downward. We say that a label l has *ordinate* y_0 if the y-coordinate of its anchor point is y_0. We assume that the labels are ordered by increasing y-coordinates.

Following Bekos et al. [3, 5] our objective is to minimize the total leader length. This minimizes not only the distance between site and label but it also minimizes the amount of non-data ink in the graphic and thus adheres to a rule of Tufte for graphical excellence [18]. We note that it suffices to minimize the total length of the vertical leader segments since the lengths of the horizontal segments are constant for a given view R and thus independent of the label placement.

To simplify the exposition, we assume that the sites are in general position. The exact meaning of the non-degeneracy will be clarified in the sequel. For each $1 \leq i \leq n$ let y_i be the y-coordinate of site p_i. Because we offer $\Omega(n \log n)$-time solutions, we can assume that the sites are ordered by their y-coordinates, i.e., $y_1 < \cdots < y_n$. Our first non-degeneracy assumption is that the y-coordinates of all sites are distinct.

In the following we introduce two extensions to standard boundary labeling: *dynamic* boundary labeling that changes with scale, and *clustered* boundary labeling, where irregular gaps between adjacent labels are allowed in order to offer more flexibility to reduce the total leader length. We also consider the combined, dynamic clustered setting.

Dynamic Boundary Labeling.

In a dynamic map, in which the user can zoom continuously, the task is to maintain an optimal boundary labeling at any scale that is reached during zooming. We want to keep the size of the labels fixed on screen. This means that as the zoom level z increases, the relative size of the labels grows, too (as they are being scaled by $1/z$ before being displayed on screen). The label height is thus a linear function of z, and for simplicity we assume that it is actually equal to z.

Our objective then is to compute an optimal boundary labeling as a function of z. Of course, as there are n labels, the full specification of the label positions would require n functions. However, as long as labels are stacked one on top of another, as soon as the position of one label at zoom level z is known, the positions of all other labels can be inferred. Thus, for a label placement with stacked labels it suffices to determine the following two functions of z: (1) $\lambda(z)$ – an index from $\{1 \dots n\}$ indicating which label has a direct leader, and (2) $s(z)$ – the placement of the $\lambda(z)$-th label. So given a set of sites P and some interval $[z_{\min}, z_{\max}]$ of zoom levels, the problem is to compute $\lambda(z)$ and $s(z)$ for all $z \in [z_{\min}, z_{\max}]$ such that at any z the labeling induced by $\lambda(z)$ and $s(z)$ has minimum total leader length. We present the solution to this problem in Section 3.

We note that if we know the position of the label stack at some zoom level z, it is straight-forward to compute the minimum total leader length for that label placement using the following observation by Bekos et al. [5, Section 3.3.4]: Connecting the i-th site to the i-th label in their respective y-orders for all $1 \leq i \leq n$ yields a length-minimal labeling with *po*-leaders.

Clustered Boundary Labeling.

We define clustered boundary labeling initially as an alternative to stacked boundary labeling in a static scenario. The requirement that all n labels form a single stack is reasonable if the labels occupy (almost) the whole right side of the view R. But if the sum of the label heights is less than the height of R it is often more desirable to allow arbitrary gaps between labels and assign label positions that better reflect the distribution of the sites, see Fig. 1c. So the problem is, given a view R of a set of n sites P with a fixed zoom level z, to place n disjoint labels of height z on the right boundary of R such that the total leader length of the induced labeling is minimized. Our algorithm to solve the clustered boundary labeling problem is given in Section 4.

Dynamic Clustered Boundary Labeling.

The most general and also the most interesting version of the problem combines the dynamic and clustered settings introduced above. So the question now is how the optimal label placement changes with the zoom level z, if arbitrary gaps between labels are allowed. Our solution decomposes the range of zoom levels into intervals, within which the clusters of the labels do not change and thus behave like single stacks of labels. For each zoom interval and each cluster in that interval, the solution must again specify two functions of z: (1) an index of a site in the cluster and (2) the placement of that site's label. We present the algorithm for the dynamic clustered version in Section 5.

Extensions.

The model described above will serve as our basic setting in Sections 3, 4 and 5, in which we present the technical details of our approach. In this setting, we tacitly assume that the screen is essentially unbounded: during the view changes, all sites remain visible on the screen and all labels fit in the vertical dimension of the screen. Section 6 extends the basic model to a more realistic scenario taking into account that the sites, as well as labels, may potentially move out of the screen limits due to zooming and panning.

2.2 Output-Sensitive Sweeps

Let's illustrate what we mean by *output-sensitive* algorithms. Consider the problem of computing intersections between a set of n line segments. Because there are $O(n^2)$ intersections, they can all be computed in $O(n^2)$ time by simply checking every pair of line segments for an intersection. Since there exist sets of line segments inducing $\Omega(n^2)$ intersections, this simple algorithm is worst-case optimal. On the other hand, if the number of intersections K is $o(n^2)$, then an algorithm, whose running time depends on K, i.e., on the complexity of the output, would be more efficient. Such algorithms are known as *output-sensitive*.

BO-Sweep.

The classical output-sensitive algorithm for computing line-segment intersections is the *Bentley–Ottmann sweep* (BO-sweep) [7, Chapter 2], which we outline next. The algorithm simulates moving a vertical *sweepline* L from $-\infty$ to $+\infty$ along the x-axis. The invariant maintained during the sweep is that all intersections to the left of L have been discovered. The sweepline *status* is a list of the segments that intersect L, ordered by the y-coordinates of the intersection points. The status of L changes at *events*, which are of two types: (1) L reaches a segment endpoint, and (2) L reaches the intersection point of two segments. The crux of the algorithm is in that the next intersection event always happens between segments that are neighbors in the sweepline status; this

allows one to spend only $O(\log n)$ time per event (after an initial $O(n \log n)$ sorting). Overall, the running time of the BO-sweep is $O(n \log n + K \log n)$, where K is the number of intersections in the instance (the output complexity).

The following observation will be useful for us in Section 5: The BO-sweep does not have to know about all the segments in advance; it is sufficient that the sweep learns about a segment once the sweepline reaches the segment's left endpoint.

Median Level in a Line Arrangement.

Let $\mathcal{H} = \{\ell_1, \ldots, \ell_n\}$ be a set of n lines in the plane. A *cell* in the *arrangement* $\mathcal{H}$ is a maximal connected component of $\mathbb{R}^2 \setminus \mathcal{H}$. Let $i \in \{1 \ldots n\}$, and let p be a point on a line ℓ in $\mathcal{H}$. The point p belongs to the *i-level* of $\mathcal{H}$ if exactly i lines (other than ℓ) lie below p. The i-level is a chain of subsegments of lines in $\mathcal{H}$; the *complexity* of the level is the number of the subsegments.

For n odd, the *median level* of $\mathcal{H}$ is the $(n-1)/2$-level. For n even, we define the median level to be the (pointwise) average of the $(n/2-1)$- and $n/2$-levels (viewing the levels as functions of x). Since the median level for an even n does not follow any of the lines of $\mathcal{H}$, we will sometimes say that it is a *dummy* level.

A long standing open question in combinatorial geometry is to bound the complexity of the i-level and, in particular, the median level [12]. The current best upper bound on the complexity of the i-level is $O(ni^{1/3})$ [9]; in particular, no better bound than $O(n^{4/3})$ is known for the complexity of the median level. The strongest lower bound on the complexity of the median level is only slightly superlinear [17].

To compute the i-level we could build the full line arrangement and trace the level. However, for lines in general position, i.e., no two lines are parallel and no three share a point, the arrangement has complexity $\Omega(n^2)$. Thus, the brute force solution will run in $\Omega(n^2)$ time. The BO-sweep is of no help here, as it would spend quadratic time computing all line intersections. A more involved sweep due to Edelsbrunner and Welzl (EW-sweep) [11], allows one to compute the i-level in a line arrangement in output-sensitive time. We describe the EW-sweep next.

EW-Sweep.

Similarly to the BO-sweep, the EW-sweep simulates sweeping a vertical sweepline L from left to right over $\mathcal{H}$, with the invariant that the level has been computed to the left of L. To bound the region in which the level changes to another line, two additional data structures are maintained during the sweep. The first data structure stores the lines that intersect L above the current edge of the level, and also the intersection of the halfplanes that are bounded from above by these lines. The boundary of this intersection defines an upper envelope of the region of interest, where the level will switch to another line. The second data structure stores analogous information for the lines intersecting L below the current edge of the level; in particular, it maintains the lower envelope of the region of interest. The lines defining the upper and lower envelopes are the candidates for becoming the next line on the level.

An *event* in the sweep involves computing the intersection of the current line supporting the level with a candidate line, and updating the two data structures. The event can be processed in $O(\log n)$ time [8, 11, 12]. Thus, the median level can be computed in $O(n \log n + K \log n)$ time, where K is the complexity of the level, i.e., the output complexity. If n is even, we can run two EW-sweeps concurrently, to compute both the $(n/2-1)$- and the $n/2$-level.

It is important for our algorithm in Section 5 that the EW-sweep develops the level "on the fly", as the sweep proceeds. In particular, suppose that at a certain point the sweep receives a signal to stop computing the level further. Then the time spent by the sweep for computing the level to the left of the current sweepline is $O(n \log n + K \log n)$, where K is the complexity of the level left of the sweepline.

3. DYNAMIC LABELING

In this section we give a solution to the dynamic boundary labeling problem, where all labels form a single stack on one side of a dynamic view R of a set P of n sites. We first assume that n is odd; then we show how to handle the case of even n.

Odd Number of Sites.

Recall that the sites $P = \{p_1, \ldots, p_n\}$ are sorted by increasing y-coordinates $y_1 < \cdots < y_n$, and that a dynamic placement of the label stack is given by functions $\lambda(z)$ and $s(z)$ – an index of some label and the position of that label at zoom level z.

The following definition is central to our algorithms:

DEFINITION 3.1. *Let $m(z)$ be the index of the median of $\{y_k - kz \mid 1 \le k \le n\}$. We call $m(z)$ the* median *of P at zoom level z. The $m(z)$-th leader connecting $p_{m(z)}$ to $l_{m(z)}$ is called the* median leader.

LEMMA 3.2. *For any zoom level z and an odd number of sites, the median leader in the optimal label placement is a direct leader.*

PROOF. The length of the horizontal part of any leader does not depend on the position of the label stack. If the lower side of the stack is aligned with the horizontal line $y = z/2$, then the k-th label has ordinate kz, $k = 1, \ldots, n$, and the vertical length of the k-th leader is $|y_k - kz|$. If the stack is shifted up by s, the total length of the leaders is $\sum_{k=1}^{n} |y_k - kz - s|$ which is minimized for $s = y_{m(z)} - m(z)z$. □

For instance, in Figs. 1a and 1b the median leaders 3 and 4 resp. are direct. Note also that the number of upward and downward leaders is the same; this is true in general, as the next corollary shows.

COROLLARY 3.3. *For any zoom level z and an odd number of sites the number of upward leaders in an optimal label placement equals the number of downward leaders.*

PROOF. Assume to the contrary that without loss of generality the number of upward leaders is larger than the number of downward leaders. As the median leader is direct, we could shift the whole label stack downwards by some $\varepsilon > 0$ and thus decrease the total leader length. This contradicts the optimality. □

For $k = 1, \ldots, n$ let $\ell_k = \{(z, y) \mid y = y_k - kz\}$ be a line in the (z, y)-plane; let $\mathcal{A}$ be the arrangement of lines $\ell_1, \ldots, \ell_n$. By definition, the median level of $\mathcal{A}$ traces the line $\ell_{m(z)}$ for any zoom level z. By Lemma 3.2, the optimal solution for the

dynamic boundary labeling problem is given by $\lambda(z) = m(z)$ and $s(z) = y_{m(z)}$. Since the EW-sweep builds the median level in time $O(n \log n)$ plus $O(\log n)$ time per edge of the level, we have

THEOREM 3.4. *The dynamic boundary labeling problem for a set P of n sites can be solved in time $O(n \log n + \sigma \log n)$ where σ is the complexity of the median level of the line arrangement $\mathcal{A}$ induced by P.*

We say that a *takeover* event happens at zoom level z if the median changes at z. For $i < j$ the median may possibly change from i to j or from j to i only at zoom level $z_{ij} = (y_j - y_i)/(j - i)$. Let $T^* = \{z_{ij} \mid z_{ij} = (y_j - y_i)/(j - i), 1 \le i < j \le n\}$ be the set of possible takeover events. Our next non-degeneracy assumption is that all events z_{ij} in T^* are distinct. In terms of the line arrangement $\mathcal{A}$ it means that all intersection points of the lines have distinct z-coordinates.

Even Number of Sites.

If n is even, the optimal label placement is not unique; to enforce uniqueness we follow the convention that the median of an even set of numbers is the mean of the two possible medians. Specifically, let $m^-(z)$, $m^+(z)$ be the indices of the two lines in the arrangement $\mathcal{A}$ that define the $(n/2-1)$- and $n/2$-levels at z. For even n the shift s (see Lemma 3.2) that minimizes the total leader length $\sum_{k=1}^{n} |y_k - kz - s|$ can be any number in the interval $[y_{m^-(z)} - m^-(z)z, y_{m^+(z)} - m^+(z)z]$. We choose s as the midpoint of the interval, i.e., $s = (y_{m^-(z)} + y_{m^+(z)})/2 - (m^-(z) + m^+(z))z/2$. This way, the $m^-(z)$-th label is at position $s^-(z) = m^-(z)z + s = (y_{m^-(z)} + y_{m^+(z)})/2 + (m^-(z) - m^+(z))z/2$, and the $m^+(z)$-th label is at position $s^+(z) = m^+(z)z + s = (y_{m^-(z)} + y_{m^+(z)})/2 + (m^+(z) - m^-(z))z/2$.

In order to have a common notion for even and odd n we introduce a *dummy site* $p_{\bar{m}(z)}$ at ordinate $(y_{m^+(z)} + y_{m^-(z)})/2$, whose dummy leader is direct. That dummy site, although not actually part of P, is called the median of $P = \{p_1, \ldots, p_n\}$. Its dummy label has height 0 so that it does not affect the placement of the real labels. Whenever either of $m^-(z)$ and $m^+(z)$ change, the median also changes. We still call the change event a *takeover*.

With the above conventions, Lemma 3.2 and Theorem 3.4 extend to the case of even n verbatim: In an optimal placement of the label stack, the median leader is direct and it can be computed as a function of z in time $O(n \log n + \sigma \log n)$.

4. CLUSTERED LABELING

In the remainder of the paper we consider the case that the labels do not have to form a single stack. Rather labels can be split into separate clusters with arbitrary gaps in between in order to reduce the total leader length. In this section we solve the problem for static boundary labeling, where the zoom level z, and thus the label height, is fixed. Without loss of generality, we assume $z = 1$ in this section. In the next section we will consider the general case of maintaining an optimal clustered labeling in the dynamic setting – for arbitrary, changing zoom level z.

We say that the sites $p_i, p_{i+1}, \ldots, p_j$ (and equivalently the labels $l_i, l_{i+1}, \ldots, l_j$) form a *cluster* C_{ij} if in the optimal solution of the clustered boundary labeling problem the labels $l_i, \ldots, l_j$ are stacked one on top of another without gaps. Let L_{ij} (resp. U_{ij}) be the y-coordinate of the lower (resp. upper) boundary of the stack. The interval $[L_{ij}, U_{ij}]$ is the answer to the following query: If $p_i, \ldots, p_j$ were the only sites (i.e., the sites $p_1, \ldots, p_{i-1}$ and $p_{j+1}, \ldots, p_n$ did not exist) and they were to from a cluster C_{ij}, what would be the extent of the cluster?

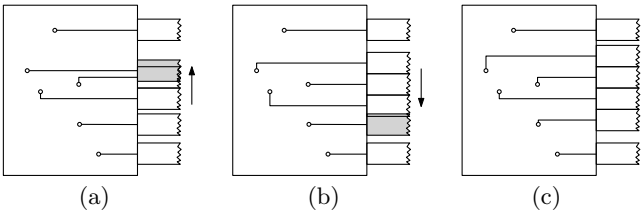

Figure 2: Illustration of the label clustering algorithm. A conflict between two clusters in an upward pass (a), in a subsequent downward pass (b), and the optimal placement (c).

Let m_{ij} be the (possibly dummy) index of the median of $\{y_i - i, \ldots, y_j - j\}$. We extend Definition 3.1 to each cluster by calling m_{ij} the cluster median of C_{ij}. For any cluster, Lemma 3.2 applies. Hence, for any cluster C_{ij}, if m_{ij} is known, one can compute L_{ij} and U_{ij} in constant time. This forms the basis of our algorithm described next.

We maintain a list $\mathcal{C}$ of clusters, initially containing just the singleton cluster C_{11}. The first step is to check whether labels l_1 and l_2 form a cluster. If $U_{11} < L_{22}$ then labels l_1 and l_2 do not overlap and both leaders are direct; they do not form a cluster and we add C_{22} on top of $\mathcal{C}$. Otherwise, they must form a cluster and we replace C_{11} by C_{12} in $\mathcal{C}$.

A generic step of the algorithm consists of *upward* and *downward* passes over $\mathbb{L}$, each of which continues until merging of clusters is dictated. During the upward pass, the cluster C_{ij} from the top of $\mathcal{C}$ is taken, and we check whether $U_{ij} \ge L_{j+1\,j+1}$, i.e., whether the labels in C_{ij} and $C_{j+1\,j+1}$ would overlap (see Fig. 2a). If yes, l_{j+1} is added to the cluster, i.e., C_{ij} is replaced with $C_{i\,j+1}$, and the upward pass continues. If no, the downward pass starts and continues until a "No" is received: it is checked whether $L_{ij} \le U_{k\,i-1}$ where $C_{k\,i-1}$ is the next-to-the-top cluster in $\mathcal{C}$, see Fig. 2b. If yes, the cluster C_{kj} is formed and replaces both $C_{k\,i-1}$ and C_{ij} in $\mathcal{C}$; the downward pass continues. Otherwise, if $L_{ij} > U_{k\,i-1}$ and the answer is "No", we start another upward pass to check for label overlaps at the upper boundary of C_{ij}. These alternating upward and downward passes continue until neither the lower nor the upper boundary of the top cluster in $\mathcal{C}$ are in conflict with the two neighboring clusters, i.e., all clusters currently in $\mathcal{C}$ are disjoint, see Fig. 2c. Then, if C_{ij} is the current top cluster, we add the new cluster $C_{j+1\,j+1}$ (which, by construction, does not intersect C_{ij}) to the top of $\mathcal{C}$ and start a new upward pass from $C_{j+1\,j+1}$.

We now analyze the total time that the algorithm spends in upward passes (the time spent in downward passes is asymptotically the same). The analysis is similar to the classical analysis of Graham's scan algorithm for computing the convex hull of a planar point set [7, Chapter 1.1]. We say that j *queries* $j+1$ when $U_{ij} \ge L_{j+1\,j+1}$ is checked. As soon as a "Yes" is received on a query, label l_j becomes internal to its cluster and j never queries any other index again. Thus, all we need to bound is the number of times

that a "No" is received on a query. Of course, j queries $j+1$ at least once. For any extra query, the cluster C of l_j must have increased during a downward pass, i.e., another cluster C' must have been merged with C. We charge that merge event to the topmost label l in C'. Since l is now internal to the merged cluster, it will never be charged again; hence the total number of queries (over all labels) is linear.

Finally, note that in order to spend $O(\log n)$ time per query, we need to update the cluster medians as the clusters merge in $O(\log n)$ time.

LEMMA 4.1. *Given a set $P = \{p_1, \ldots, p_n\}$ of n sites ordered by y-coordinates, a data structure can be built in $O(n \log n)$ time that allows one to do median queries for clusters C_{ij} in $O(\log n)$ time.*

PROOF. The data structure for answering cluster median queries is a modified layered range tree using fractional cascading [7, Chapter 5.6]. We first transform the set of sites P into a set of points $Q = \{(y_i - i, i) \mid 1 \le i \le n\}$. For a cluster C_{ij} we are interested in the median m_{ij} of the set $\{y_k - k \mid i \le k \le j\}$. That is, we need to find the median of the first coordinate of the points in Q that are in the query region $(-\infty, \infty) \times [i, j]$.

We build a binary search tree $\mathbb{T}$ in the first coordinate of Q, where each node v in $\mathbb{T}$ holds an associated array $A(v)$ of those points of Q whose first coordinate is in the subinterval represented by v. These arrays are sorted in increasing order by the second coordinate of Q. Let v be an internal node of $\mathbb{T}$ and let $\mathrm{lc}(v)$ and $\mathrm{rc}(v)$ be the left and right children of v. Each cell in $A(v)$ stores a pointer into $A(\mathrm{lc}(v))$ and a pointer into $A(\mathrm{rc}(v))$. Each pointer is actually the index of the smallest element in $A(\mathrm{lc}(v))$ (resp. $A(\mathrm{rc}(v))$) larger or equal to the element in the cell of $A(v)$ (if such an element exists).

We can use the pointers in the associated arrays to locate the k-th largest element (and thus the median) in the query region for cluster C_{ij} as follows. We locate the elements corresponding to sites p_i and p_j in the array $A(r)$ of the root r of $\mathbb{T}$ in constant time as they are in the i-th and j-th cell of $A(r)$, respectively. Then we descend into the left child of r and follow the pointers in the i-th and $(j+1)$-th cells to locate the boundaries of the interval of points in $A(\mathrm{lc}(r))$ that fall into the query region. The pointers are actually two array indices $l_{\mathrm{lc}(r)}$ and $u_{\mathrm{lc}(r)}$ and thus the number of elements in the query region represented by $\mathrm{lc}(r)$ is $a = u_{\mathrm{lc}(r)} - l_{\mathrm{lc}(r)}$. If $u_{\mathrm{lc}(r)} - l_{\mathrm{lc}(r)} < k$ then we know that the k-th largest element represented by r is the $(k-a)$-th largest element represented by $\mathrm{rc}(r)$; we continue the search in $\mathrm{rc}(r)$ accordingly. Otherwise the k-th largest element is in the current subtree and we recursively continue our search in the left child of $\mathrm{lc}(r)$.

Since the height of $\mathbb{T}$ is $O(\log n)$ the query time of $O(\log n)$ follows immediately. The size of $\mathbb{T}$ and its preprocessing time is $O(n \log n)$ analogous to standard layered range trees [7, Chapter 5.6]. □

Our algorithm in combination with Lemma 4.1 yields

THEOREM 4.2. *The clustered boundary labeling problem for a set P of n sites and a fixed zoom level z can be solved in $O(n \log n)$ time.*

The correctness of the algorithm is established via the following invariant that holds by construction after each generic step of the algorithm: If C_{ij} is the top cluster in $\mathcal{C}$, then, if sites $p_{j+1}, \ldots, p_n$ did not exist, the labels $l_i, \ldots, l_j$ are placed optimally (even) ignoring the sites $p_1, \ldots, p_{i-1}$, and the labels in the other clusters in $\mathcal{C}$ are placed optimally ignoring sites $p_i, \ldots, p_j$.

5. DYNAMIC CLUSTERED LABELING

In this section we consider the most general problem that combines dynamic and clustered boundary labeling. We show that the solution to the problem can be represented by a tree $\mathcal{T}$ in the arrangement $\mathcal{A}$ of lines $\{\ell_1, \ldots, \ell_n\}$ defined as in Section 3. We build the tree in time $O(n \log^2 n + \gamma \log n)$ where γ is the combinatorial complexity of $\mathcal{T}$.

We construct the labeling with increasing zoom level z. Initially, for $z = 0$, every leader is a direct leader and all labels form singleton clusters. With growing label size, clusters will merge and eventually all labels will form a single cluster. Our first observation is that the clusters are monotone in the following sense:

LEMMA 5.1. *Let C_{ij} be a cluster of labels $l_i, \ldots, l_j$ appearing in an optimal clustered labeling at some zoom level z_0. Then for any zoom level $z \ge z_0$ the labels $l_i, \ldots, l_j$ remain in a joint cluster in the optimal clustered labeling.*

PROOF. In order to show this lemma we observe that each cluster is held together by a "gravity" effect. Let $P_{ij} = \{p_i, \ldots, p_j\}$ be the sites connected to the labels in C_{ij}. By Lemma 3.2 and Corollary 3.3 we know that the median leader of P_{ij} is direct and that there is an equal number of upward and downward leaders from P_{ij}. We show that for any index k with $i \le k < j$ at most half of the leaders for sites in the set P_{ik} are upward and at most half of the leaders for sites in the set $P_{k+1\,j}$ are downward. So shifting the labels $l_i, \ldots, l_k$ downwards or shifting the labels $l_{k+1}, \ldots, l_j$ upwards does not decrease the total leader lengths of P_{ik} and $P_{k+1\,j}$. This implies that the total leader length of P_{ij} cannot be decreased by splitting C_{ij} between l_k and l_{k+1}. Hence, C_{ij} remains a single cluster in the optimal labeling for $z \ge z_0$ until it is eventually merged with a neighboring cluster. From that point on the same argument holds for the merged cluster.

At zoom level z_0 clearly at most half of the leaders for sites in the set P_{ik} are upward for any $i \le k < j$; otherwise we could shift the labels $l_i, \ldots, l_k$ downwards by some $\Delta > 0$ and decrease the total leader length. This contradicts the assumption that C_{ij} is a cluster in the optimal labeling for zoom level z_0. An analogous argument holds for the number of downward leaders of sites in $P_{k+1\,j}$.

If we let z grow from some zoom level $z_1 \ge z_0$ for which the above property still holds we need to distinguish two cases. If the current cluster median is contained in $P_{k+1\,j}$ then the anchor points of the labels for P_{ik} are all moving downwards and the number of upward leaders obviously does not grow. If, on the other hand, the current median is some site $p_{k'}$ in P_{ik} then the vertical segments of all downward leaders for sites in P_{ik} above the median actually shrink. But whenever such a leader for a site p_m $(k' < m \le k)$ becomes a direct leader at some zoom level z_2, a takeover event takes place. The direct leader of p_m becomes the new median leader and the previous median leader, since it belongs to a site below the new median, turns into a downward leader. Hence the number of upward leaders of sites in P_{ik} does not grow regardless of the cluster median. A symmetric

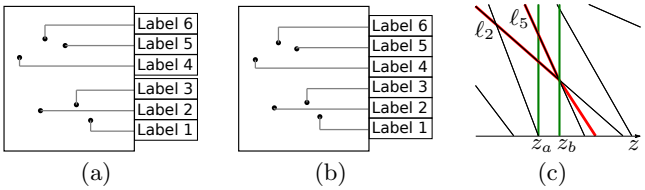

Figure 3: Two clusters (a) before and (b) after a merge event; (c) embedding of the merge tree. Points z_a and z_b in (c) represent the zoom levels of figures (a) and (b), respectively. For the merge event at zoom level z_b the clusters with median sites p_2 and p_5 merge, and the lines ℓ_2 and ℓ_5 in (c) intersect.

argument holds for the number of downward leaders of any set $P_{k+1\,j}$. □

We say that a *merge event* happens when two clusters merge. Between two merge events, each cluster behaves as a separate stack of labels to which the results of Section 3 apply. In particular, the median level in each cluster can be built independently by an EW-sweep. We use a separate EW-sweepline for each cluster, and also a global event queue in which the events from all EW-sweeps are stored. In $O(\log n)$ time per EW-event (i.e., a takeover event in any of the EW-sweeps), both the status of the relevant EW-sweepline and the global event queue can be updated.

Thus, the median levels within all clusters can be built in $O(\gamma \log n)$ time where γ is the total complexity of the medians in the clusters. What remains to show is how to detect and handle the merge events.

It appears that a merge event for two clusters occurs at the time when their median levels intersect (Fig. 3). Indeed, suppose that both clusters have an odd number of labels, and let $i < j$ be the two cluster medians. As the zoom level increases, the clusters "collide" at zoom level z when the distance between their medians equals the number of labels between them, i.e., $(j-i)z = y_j - y_i$, which is exactly the zoom level at which the lines ℓ_i and ℓ_j intersect in $\mathcal{A}$. Using dummy medians, the same is true when two even clusters or two different-parity clusters merge.

Thus, to detect merge events it is enough to watch for intersections between the medians during the sweep. For that we perform, concurrently with the EW-sweeps, an additional BO-sweep whose sweepline is called the *median sweepline* (see Fig. 4). The status of the median sweepline is the list of the cluster medians in the order as they intersect the sweepline. The events are intersections between neighboring medians. The events are stored in the same, global queue. Every time a takeover event happens in any of the EW-sweeps, a signal is sent to the BO-sweep to update the status of the median sweepline and the event queue of the BO-sweep; each update takes $O(\log n)$ time.

Finally, we describe what happens to the EW-sweeps at a merge event in the BO-sweep. Recall that the following information is stored with an EW-sweepline: the list of lines currently above (resp. below) the median and the intersection of the halfplanes bounded by these lines. Suppose now that two clusters C and C' merge. The median for the new cluster is chosen from the following constant-size set of candidates (which depends on the parity of $|C|$ and $|C'|$): one of the current medians of C and C' or a new dummy median. Next,

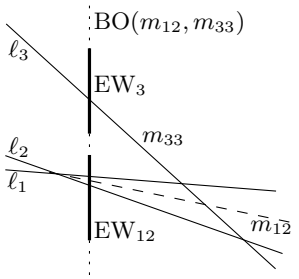

Figure 4: Two EW-sweeps (one per cluster) simultaneously "develop" the cluster medians m_{12} (dummy) and m_{33}. The medians from *all* current clusters are the segments in the global BO-sweep that detects the median intersections, i.e., the cluster merge events.

we take the lines stored with the EW-sweepline of the smaller cluster and add them one by one to the EW-sweepline of the larger cluster. Note that the lines below (above) the medians in C or C' are also below (above) the median in the new cluster – modulo the fact that depending on the parities of $|C|$ and $|C'|$ the previous median lines may change their status to being above or below the new (dummy) median. For every added line, we update the halfplane intersections; this can be done in $O(\log n)$ time per addition of a line [8].

The Merge Tree.

We now bound the total time spent in the merge events. By Lemma 5.1 there are exactly $n-1$ merge events for $z \in (0, \infty)$. Moreover, the set of all clusters forms a hierarchical decomposition of the sites, i.e., the formation of clusters with increasing zoom level can be represented by a tree T. The leaves of T are individual sites, and internal nodes represent the merging of the children clusters into a new cluster; by our non-degeneracy assumption no two merge events happen simultaneously and thus the tree is binary. The time our algorithm spends at a node v of T is $O(s_{\min}(v) \log n)$ where $s_{\min}(v)$ is the minimum of the sizes of the two subtrees of v, and the total time spent to process all $n-1$ merge events and to update the status of all EW-sweeplines is $O(\sum_{v \in T} s_{\min}(v) \log n)$.

In the next lemma we prove that $\sum_{v \in T} s_{\min}(v)$, i.e., the sum of the sizes of the smaller subtrees of the nodes of T, is $O(n \log n)$. As an immediate consequence, we obtain that the total time spent by our algorithm to process the merge events is $O(n \log^2 n)$.

LEMMA 5.2. *The sum* $\sum_{v \in T} s_{\min}(v)$ *is* $O(n \log n)$.

PROOF. The *heavy-path decomposition* (HPD) [13] of a tree T is a decomposition of T into paths as follows. Let u be a node of T and let T_u be the subtree rooted at u. Then a child w of a non-leaf node u is called the *heavy child* of u if $|T_w| \geq |T_v|$ for all children v of u. All other children of u are *light children*. In case of a tie, we arbitrarily designate one node as the heavy child of u. An edge of T is called *heavy* if it connects a heavy child to its parent; all other edges are *light*. The heavy edges induce a set of pairwise-disjoint heavy paths in T that, together with the remaining leaves as single-vertex heavy paths, form the HPD (see Fig. 5). Each node of T belongs to exactly one heavy path. The *decomposition tree* $H(T)$ has the heavy paths as nodes and the light edges as edges between two heavy paths.

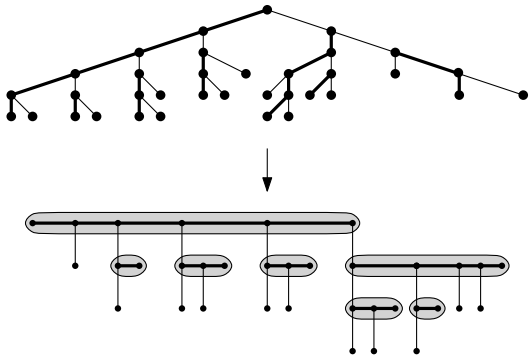

Figure 5: The heavy path decomposition of a binary tree T (top) and the corresponding decomposition tree $H(T)$ (bottom). Heavy edges are shown in bold. Light subtrees are subtrees of nodes whose parent edge is a light edge.

With the above definitions, $\sum_{v \in T} s_{\min}(v)$ is the sum of the subtree sizes of the light children over all nodes of T. By definition of the HPD, a node u is the light child of some node v if and only if u is the root of a heavy path (but not the root of T). Thus the sum of the subtree sizes of the light children equals the total size of the subtrees rooted at the roots of heavy paths (except the heavy path at the root of $H(T)$).

For a heavy path π let the *light subtrees* of π be the subtrees rooted at the light children of nodes of π. The light subtrees of π are pairwise-disjoint. Moreover, let π and π' be heavy paths at the same level of $H(T)$ (a level in a tree is the set of nodes at the same distance to the root). Clearly, the light subtrees of π are disjoint from the light subtrees of π'. Thus, the total size of the light subtrees of heavy paths at any one level of $H(T)$ is $O(n)$. Harel and Tarjan [13] showed that $H(T)$ has $O(\log n)$ levels. Thus the total size of the light subtrees of all heavy paths is $O(n \log n)$, and the lemma follows. □

Interestingly, the merge tree T has a natural embedding $\mathcal{T}$ in the arrangement $\mathcal{A}$ (augmented with dummy lines where necessary). The embedding $\mathcal{T}$ is the union of the median levels of the clusters. An internal node of T corresponds to a degree-3 node of $\mathcal{T}$; an edge of T between a child u and its parent v is a path in $\mathcal{T}$, namely the median level of the cluster corresponding to u. In Fig. 6 one can see the embedding of the merge tree produced by our implementation of the algorithm (more details on the implementation are provided in the next section).

The combined BO- and EW-sweeps described above build the median levels for all clusters, and hence also the tree $\mathcal{T}$. Overall, this leads to our main result summarized in the following theorem:

THEOREM 5.3. *The tree $\mathcal{T}$ that describes the optimal clustered label placement for all zoom levels $z \in (0, \infty)$, can be built in $O(n \log^2 n + \gamma \log n)$ time, where γ is the combinatorial complexity of $\mathcal{T}$.*

6. PRACTICAL MATTERS

An important practical aspect of dynamic maps is that the display area of a screen or a mobile device is limited. Our algorithms in the previous sections were presented without taking the limited area of the rectangular view R into account. In this section, we discuss how the algorithms can be adapted to produce boundary labelings for map views limited to a screen of width w_s and height h_s. Furthermore, we report on the implementation of our algorithms as a Java applet showing their relevance in practice.

6.1 Extensions of the Model

Any particular view R, whose aspect ratio r matches the aspect ratio w_s/h_s of the screen can be expressed by three *extended world coordinates* introduced by Been et al. [2] for dynamic map labeling. The coordinates define a three-dimensional (x, y, z)-space where the x- and y-coordinates are a position in the map plane, and the z-coordinate is a zoom level. The view $R = R(x, y, z)$ is then the axis-aligned rectangle in the map plane with the bottom-left corner (x, y), width $w_s z$, and height $h_s z$. User interaction (zooming and panning) over time can now be easily expressed as a trajectory of the view in extended world coordinates.

Visible Sites.

The first effect of displaying only a limited view R of the map is that the set of sites, visible in R, changes as the user pans and zooms the map. Let $p_i = (x_i, y_i)$ be a site in P. The site is visible in a view $R(x, y, z)$ if and only if the following four inequalities hold:

$$\begin{array}{rcccl} x_i - w_s z & \leq & x & \leq & x_i \\ y_i - h_s z & \leq & y & \leq & y_i. \end{array} \tag{1}$$

The inequalities (1) are linear in x, y, and z. Each inequality defines a halfspace bounded by a plane in the extended world coordinates. Every view that corresponds to a point in the intersection of these four halfspaces contains the site p_i. Let $\mathcal{V}$ be the arrangement of the $4n$ planes induced by all n sites $P = \{p_1, \ldots, p_n\}$; the complexity of $\mathcal{V}$ is $O(n^3)$. Every point (x, y, z) within one cell of $\mathcal{V}$ defines a view $R(x, y, z)$ with the same set of visible sites. In particular, we can apply our algorithms in each of the cells to have the optimal label placement ready for any possible view that the user might choose. As we track the user's interaction trajectory in extended world coordinates we can detect when the trajectory crosses the boundary of the current cell in $\mathcal{V}$. At this point, we simply switch to the solution for the adjacent cell in $\mathcal{V}$.

Visible Labels.

The bounded area of the view also has an effect on the label placement, as the user zooms and pans, even when the set of visible sites does not change. Obviously, for a given label height h_l on screen, not more than $\lfloor h_s/h_l \rfloor$ labels can be placed along the boundary. Furthermore, when labels move vertically – either due to panning or in order to maintain their optimal positions according to our algorithms during zooming – they might move beyond the boundaries of the screen. To avoid this, motion of a label should be stopped as soon as it reaches the top or bottom screen boundary. Next, we describe how to do this for the upper boundary of the screen; the situation with the lower boundary is symmetric.

The ordinate of the upper boundary of the view $R(x, y, z)$ is $u_R(x, y, z) = y + h_s z$. Let C be the topmost cluster of labels. Our solution from the previous section specifies, for any zoom level z, an index of a site from C and the placement of the label for that site. This allows us to calculate the ordinates of the upper and lower boundaries of C as

two functions $u_C(z)$ and $l_C(z)$ of the zoom level z. When following the user's interaction trajectory we keep track of $u_R(x, y, z)$ and $u_C(z)$. Whenever $u_C(z) > u_R(x, y, z)$ we switch to a different solution for the placement of cluster C by setting the upper boundary of the label stack to $u_R(x, y, z)$. Knowing the number of labels in C and the zoom level z we can immediately compute the new corresponding lower boundary function $l'_C(z)$. Just as the topmost cluster C observes the upper boundary $u_R(x, y, z)$, any other cluster C' observes the actual lower boundary $l'_{C''}(z)$ of its upper neighbor C'' in an analogous way. (Note that if C'' was not involved in a collision with a cluster or screen boundary, then $l'_{C''}(z) = l_{C''}(z)$; otherwise $l'_{C''}(z)$ is computed based on the collision.)

When panning the view, it can also happen that the original upper boundary of some cluster C moves below the upper screen boundary (or the lower boundary of the cluster C' above it), i.e., $u_C(z) \leq u_R(x, y, z)$ (or $u_C(z) \leq l'_{C'}(z)$). In this case we simply switch back to the original placement of C computed by the algorithm from the previous section.

6.2 Implementation

We implemented our dynamic boundary labeling algorithms as a Java applet, which is available (including the source code) at `http://cs.helsinki.fi/group/compgeom/boundarylabeling/`. Figure 6 shows a screenshot. The implementation allows the user to add sites to the map manually or randomly and to assign site priorities. The latter feature is used to select the K most important sites to be labeled if only K labels are available, e.g., to avoid placing more than the maximum possible number $\lfloor h_s/h_l \rfloor$ of labels, or to avoid clutter on screen. The user can specify the maximum number of visible labels.

Our implementation does handle the two extensions of the model for visible sites and visible labels described in Section 6.1. For visible sites, however, we currently do not compute the arrangement of planes in the extended world coordinates. Rather we recompute the line arrangement every time a site enters or leaves the current view. Also, instead of implementing the EW-sweep with the optimal but complicated $O(\log n)$-time dynamic convex hull algorithm [8], we use a simpler convex hull structure with $O(n)$ query time. Our experimental results below and hands-on experience with the applet show that despite the recomputations and the straight-forward way to build the tree, our simple approach is still fast enough for real-time map labeling.

Experimental Results.

We measured the performance of the implementation to see whether it is fast enough in practical cases. Even for random inputs with about a thousand sites the merge tree generation takes only a few tens of milliseconds on a standard PC with 2.13Ghz processor. So there is no visible delay in our interactive application.

To get an idea of the problem complexity in practice, we took for every n from 1 to 4000 the average complexity of the median level and merge tree over 5 instances of n sites distributed uniformly at random in the rectangular viewing area. Both σ and γ exhibited very slightly superlinear growth with relatively small deviations in our tests, see Fig. 7. Due to the dummy medians, the complexity of the median level for even n is twice as large as that for odd n (this is the reason why σ appears as two lines on the plot).

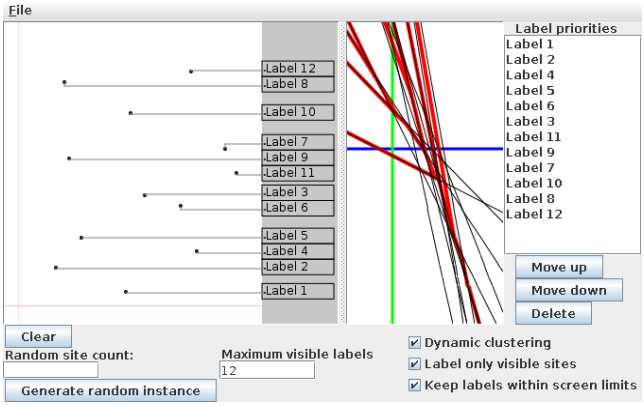

Figure 6: A screenshot of our implementation. The left pane of our applet is the map; the right pane shows the line arrangement $\mathcal{A}$ and highlights the merge tree $\mathcal{T}$ in red.

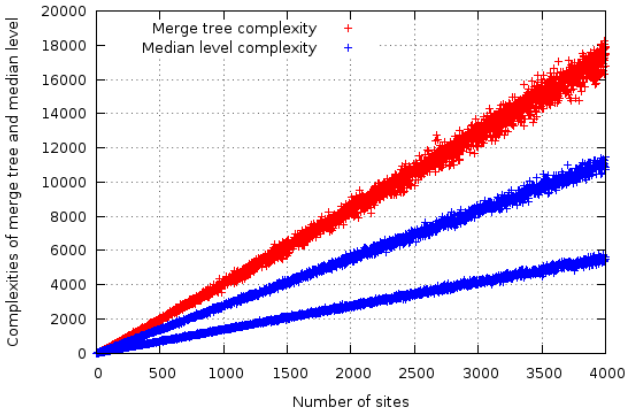

Figure 7: Average complexity σ of the median level (blue) and γ of the merge tree $\mathcal{T}$ (red) for random inputs.

7. DISCUSSION

In this paper we introduced the dynamic (one-sided) boundary labeling problem as an extension of the static boundary labeling problem. We designed, analyzed, and implemented efficient algorithms to compute and maintain clustered and non-clustered optimal label placements during user interactions that are composed of zooming and panning a map view.

Our algorithms for dynamic labeling are output-sensitive: their running times are a sum of the preprocessing time ($O(n \log n)$ or $O(n \log^2 n)$) and a term that depends on the output complexity (σ – the complexity of the median level in the line arrangement $\mathcal{A}$, and γ – the complexity of the merge tree $\mathcal{T}$). The dependence on the complexities is reasonably good: we spend only $O(\log n)$ time for each "breakpoint" of the solution. Our experiments suggest that the complexities are almost linear in practice, but it would still be interesting to give *worst-case* upper bounds on σ and γ.

In general, the best bounds on the complexity of the median level in a line arrangement are currently $\Omega(n^{1+\varepsilon})$ [17] and $O(n^{4/3})$ [9]. But the arrangement $\mathcal{A}$ has the special property that the slopes of the lines are consecutive integers. Edelsbrunner et al. [10] present a lower bound of $\Omega(n \log n)$

on the number of halving lines for points with consecutive integer abscissas; by the duality of the median level in a line arrangement and the set of halving lines in a point set, the complexity σ of $\mathcal{A}$ can thus be $\Omega(n \log n)$, too. Can Tóth's [17] construction of the $\Omega(n^{1+\varepsilon})$ lower bound on the number of halving lines be adapted so that the points' abscissas are $\{1, \ldots, n\}$? On a similar note, it would be interesting to have bounds on the combinatorial complexity of our merge tree $\mathcal{T}$.

In our extended dynamic boundary labeling model (Section 6.1), we define a three-dimensional arrangement $\mathcal{V}$ of planes. We observed that the labeling solutions for all map views represented by the points in one cell of $\mathcal{V}$ are based on the same line arrangement $\mathcal{A}$ or merge tree $\mathcal{T}$, respectively. It is an interesting open question whether we can precompute $\mathcal{A}$ and $\mathcal{T}$ for all $O(n^3)$ cells of $\mathcal{V}$ more efficiently than by applying our algorithms separately for each cell.

One of the constraints in *static* boundary labeling is to find labelings, in which no two leaders intersect. It is thus a natural and open question to ask for crossing-free *dynamic* labelings. A major concern with dynamic crossing removal is, however, that the vertical order of the labels is modified every time a crossing is removed. This may lead to frequent and rather drastic changes of the labeling, to which the user has to readjust mentally at every such event. Recall that (dynamic) crossing removal and leader length minimization are independent problems since the label positions and the total leader length are not affected by removing crossings [5].

We suggest the following semi-dynamic compromise to balance the trade-off between the visual quality of a snapshot of the labeling at any time and the visual quality of the whole animation during user interaction. Instead of requiring that every frame of the animation must be a crossing-free labeling, we remove crossings only when the user stops moving the map view; this can be done in $O(n \log n)$ time [6]. Upon resumption of the movement, we keep the current order of the labels until the next break. To support the preservation of the mental map during crossing removal, the swapping of label positions can be animated so that all changes are continuous.

Another open problem is two- or four-sided dynamic boundary labeling, which could in general yield labelings with shorter leaders and more or larger labels. One advantage of one-sided dynamic labelings, however, is that they preserve the user's mental map better than multi-side labelings, in which labels could frequently switch sides during user interaction. As a compromise, we suggest a similar semi-dynamic approach as for crossing removal. Every time the user stops moving, we can apply the existing algorithms for static multi-side boundary labeling [5] to compute an optimal solution. During user interaction, however, we stick to the existing assignment of the labels to the boundaries of the map view, and apply our one-sided algorithms separately to each boundary side of the labeling.

Acknowledgments. We thank Jie Gao, Alon Efrat, David Eppstein and Jukka Suomela for discussions and the anonymous reviewers for helpful suggestions. This research was supported in part by the German Research Foundation under grant NO 899/1-1 and by Academy of Finland grant 118653 (ALGODAN).

8. REFERENCES

[1] K. Been, E. Daiches, and C. Yap. Dynamic map labeling. *IEEE Trans. Visualization and Computer Graphics*, 12(5):773–780, 2006.

[2] K. Been, M. Nöllenburg, S.-H. Poon, and A. Wolff. Optimizing active ranges for consistent dynamic map labeling. *Comput. Geom. Theory & Applications*, 43(3):312–328, 2010.

[3] M. Bekos, M. Kaufmann, M. Nöllenburg, and A. Symvonis. Boundary labeling with octilinear leaders. *Algorithmica*, 57(3):436–461, 2010.

[4] M. Bekos, M. Kaufmann, K. Potika, and A. Symvonis. Multi-stack boundary labeling problems. In *Proc. Found. Softw. Technol. and Theor. Comput. Sci. (FSTTCS'06)*, *LNCS* 4337, pp. 81–92, Springer, 2006.

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

[6] M. Benkert, H. Haverkort, M. Kroll, and M. Nöllenburg. Algorithms for multi-criteria boundary labeling. *J. Graph Algorithms Appl.*, 13(3):289–317, 2009.

[7] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2008.

[8] G. S. Brodal and R. Jacob. Dynamic planar convex hull. In *Proc. 43rd Symp. Foundations of Comput. Sci. (FOCS 2002)*, pp. 617–626, 2002.

[9] T. K. Dey. Improved bounds for planar k -sets and related problems. *Discrete Comput. Geom.*, 19(3):373–382, 1998.

[10] H. Edelsbrunner, P. Valtr, and E. Welzl. Cutting dense point sets in half. *Discrete Comput. Geom.*, 17(3):243–255, 1997.

[11] H. Edelsbrunner and E. Welzl. Constructing belts in two-dimensional arrangements with applications. *SIAM J. Comput.*, 15:271–284, 1986.

[12] D. Halperin. Arrangements. In J. E. Goodman and J. O'Rourke, eds., *Handbook of Discr. and Comput. Geom.*, Ch. 24, pp. 529–562, 2004.

[13] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

[14] M. Kaufmann. On map labeling with leaders. In S. Albers, H. Alt, and S. Näher, editors, *Efficient Algorithms*, *LNCS* 5760, pp. 290–304, Springer, 2009.

[15] I. Petzold. *Beschriftung von Bildschirmkarten in Echtzeit*. PhD thesis, Universität Bonn, 2003.

[16] I. Petzold, G. Gröger, and L. Plümer. Fast screen map labeling—data-structures and algorithms. In *Proc. 23rd Intl. Cartographic Conf. (ICC'03)*, pp. 288–298, 2003.

[17] G. Tóth. Point sets with many k-sets. *Discrete Comput. Geom.*, 26(2):187–194, 2001.

[18] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 2nd edition, 2001.

[19] M. v. Kreveld. Geographic information systems. In J. E. Goodman and J. O'Rourke, eds., *Handbook of Discr. and Comput. Geom.*, Ch. 58, pp. 1293–1314, 2004.

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