Dynamic Map Labeling

Ken Been, *Member, IEEE*, Eli Daiches, and Chee Yap, *Senior Member, IEEE*

Abstract— We address the problem of filtering, selecting and placing labels on a dynamic map, which is characterized by continuous zooming and panning capabilities. This consists of two interrelated issues. The first is to avoid label popping and other artifacts that cause confusion and interrupt navigation, and the second is to label at interactive speed. In most formulations the static map labeling problem is NP-hard, and a fast approximation might have $O(n \log n)$ complexity. Even this is too slow during interaction, when the number of labels shown can be several orders of magnitude less than the number in the map. In this paper we introduce a set of desiderata for "consistent" dynamic map labeling, which has qualities desirable for navigation. We develop a new framework for dynamic labeling that achieves the desiderata and allows for fast interactive display by moving all of the selection and placement decisions into the preprocessing phase. This framework is general enough to accommodate a variety of selection and placement algorithms. It does not appear possible to achieve our desiderata using previous frameworks.
Prior to this paper, there were no formal models of dynamic maps or of dynamic labels; our paper introduces both. We formulate a general optimization problem for dynamic map labeling and give a solution to a simple version of the problem. The simple version is based on label priorities and a versatile and intuitive class of dynamic label placements we call "invariant point placements". Despite these restrictions, our approach gives a useful and practical solution. Our implementation is incorporated into the G-Vis system which is a full-detail dynamic map of the continental USA. This demo is available through any browser.

Index Terms—Map labeling, dynamic maps, human-computer interface, label placement, label selection, label filtering, label consistency, computational cartography, GIS, HCI, realtime, preprocessing

1 Introduction

In computational map labeling [27, 30] the issue is to apply general cartographic principles for map labels in an efficient and automatic fashion. In most algorithmic treatment, label placement is posed as a global optimization [16]—e.g., place the maximum number of labels without overlap, given a set of alternative locations for each label. Such problems are typically NP-hard (e.g., [17, 11]) and approximations or heuristics become essential. In such a context, an $O(n \log n)$ solution would be considered very fast, although most solutions are considerably worse [16].

In this paper we are interested in **dynamic maps**, which are characterized by support for **continuous zoom** (change of scale) and **continuous pan** (change of region of interest). Labels in dynamic maps must be placed at interactive speed. Since the number of labels in the entire map can be several orders of magnitude more than the number that will be shown in the current view area, even "fast" $O(n \log n)$ static solutions are inadequate. (In our map of the USA we have over 12 million labels.) Furthermore, static maps and dynamic maps play different roles for users. A primary goal with static maps is to maximize information content. For a limited edition tricentennial anniversary map of colonial America, we should try to find a globally optimal label placement. A basic purpose of dynamic maps, on the other hand, is *navigation*, a term we use to capture a variety of tasks, such as searching for a particular location or for the most scenic route between two points. In such tasks labels are useful as navigation markers as well as for information content. For markers in a dynamic environment it is crucial to avoid behavior that is distracting or jarring, such as labels popping or moving about in unexpected ways. In other contexts researchers have sought "frame-coherency" [2] or "temporal continuity" [7]. We use the term **consistency** to capture these ideas and more.

Let's informally consider four desiderata for dynamic label consistency. As far as we know, no previous work has achieved these desiderata. In the following, **placement** refers to the location, size and orientation chosen for a label, and **selection** refers to the decision whether to show a label or not.

(D1) *Except for sliding in or out of the view area, labels should not vanish when zooming in or appear when zooming out.* This captures the usual expectation that strictly more features get labeled as we zoom in, and strictly fewer features get labeled as we zoom out. It ensures that labels do not appear, disappear and then re-appear under monotonic zooming. This is similar to the "monotonicity property" in [28, 26]. In some cases we prefer a less restricted version of this desideratum, where we might want, for example, a country label to disappear as we zoom in to street level detail.

(D2) *As long as a label is visible, its position and size should change continuously under the pan and zoom operations.*

(D3) *Except for sliding in or out of the view area, labels should not vanish or appear during panning.* If (D2) is satisfied, then (D3) can be satisfied by making the decision to select a label a function of scale.

(D4) *The placement and selection of any label is a function of the current map state (scale and view area).* In particular, it does not depend on the history of how we arrived at that state. As will be described later, under our algorithm placement and selection are functions of scale alone. In that case, the dynamic map *appears* like a collection of statically generated maps, one for each scale, even though the labeling is in fact dynamically computed.

Let's see how these rules can be violated with a naive dynamic labeling solution. As noted above, the number of labels (or the number of features to be labeled) in the entire map is much larger than the number in the current view area. So for interactive speed we need to introduce a **label filtering** step, which quickly reduces the number of labels that must be considered. For example, we could retrieve just those labels that intersect the current view area, or we could drop all local street names if we are zoomed out far enough. Our naive solution becomes this: do the label filtering, and then run a static labeling algorithm on the reduced set of labels. Figure 1 shows how desiderata (D1) and (D3) can be violated with this approach. Notice that imposing a global priority order on the labels doesn't help. The source of the problem—the fundamental stumbling block for dynamic labeling—is that from frame to frame the labeling is done on a different set of labels and/or a different set of label conflicts.

Contributions of this paper. In this paper we present a dynamic labeling solution that satisfies our consistency desiderata, operates at interactive speeds, and generates a high quality labeling. An essential component of our solution is a dynamic labeling *framework* that guar-

• *Ken Been is with Yeshiva University, E-mail: kbeen@yu.edu.*
• *Eli Daiches is with Yeshiva University, E-mail: daiches@yu.edu.*
• *Chee Yap is with New York University, E-mail: yap@cs.nyu.edu.*

Manuscript received 31 March 2006; accepted 1 August 2006; posted online 6 November 2006.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

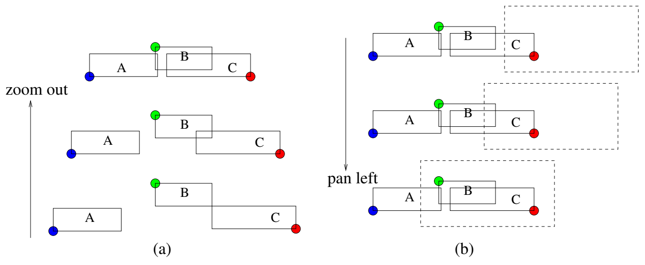


Fig. 1. Violations of (D1) under zooming in (a), and (D3) under panning in (b). In both cases, label priority is $A > B > C$. Initially C is visible. As we zoom out/pan left, C disappears because it conflicts with B. Then C reappears when B disappears because of its conflict with A.

antees consistency and interactive speed, and that is general enough to incorporate many different algorithms for the actual selection and placement of labels.

What is this framework? The issue is to solve the three problems of filtering, selection and placement of labels at interactive speeds, supported by preprocessed data structures. Conventionally, it seems best to first filter the labels to a reasonably small set (based on region of interest and/or scale), then select a subset of these labels, and finally place the selected labels. This was Petzold's approach (see Section 2). But our above examples suggest that it would be difficult to achieve consistency this way, even with label priorities. Indeed, our solution uses an "inverted sequence": we first place all the labels, then select from the placed labels. These steps are done in the preprocessing stage. Finally we use filtering during the interaction phase, to merely retrieve the precomputed selection and placement.

This unconventional sequence leads to a useful and practical dynamic labeling solution. It does, however, limit the scope of our solution to labels that don't "slide around" during panning. This is because we determine, during the preprocessing phase, a placement for each label that is a function of scale. So, for example, a label for a long road cannot slide along the road to stay within the view area as the user pans. Under our approach we can give the illusion of a single static map at each scale, and we (subjectively) see this as adequate, or even preferable, in a typical dynamic map. We can handle long roads by placing several labels along the road—precisely what would be done in a high quality paper map.

Two screenshots of our implementation appear in Figure 2, showing the same location at two different scales. This implementation is incorporated into our open-source G-Vis System [31, 5], a full detail dynamic map of the continental USA based on the publicly available Tiger data [25]. Our labeling demo is available on the internet [13].

Prior to this paper, there were no formal models of dynamic maps, nor of dynamic labels. In order to initiate a systematic study, we introduce simple versions of these concepts. We formulate a dynamic labeling optimization problem, and show that a natural version of it is NP-complete. However, we give an efficient solution to a version of the problem that relies on label priorities. We further introduce a simple but versatile class of dynamic label placements called **invariant point placements**.

In the next section we will review the related work in this area. Section 3 introduces our dynamic labeling framework, and Section 4 gives a formal model. Section 5 covers some practical considerations for implementing our approach, and we conclude in Section 6. In the appendix we prove the NP-completeness of the simple dynamic labeling optimization problem.

2 Related Work

Cartographic principles for labeling maps are laid out in [15, 32]. Starting in the 1980's, static label placement algorithms began to appear [14, 1, 12, 8, 9, 29, 16]. In the 1990's a number of techniques were developed for interactive labeling; a survey of this work was given by

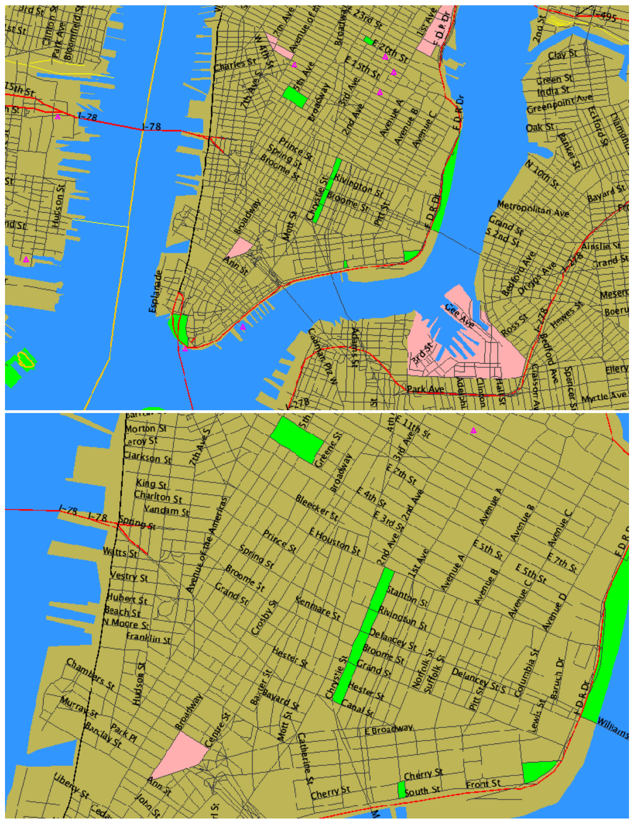


Fig. 2. New York street map, zoomed out and zoomed in

Fekete and Plaisant[10]. For example, with excentric labeling [10], all features in a circular neighborhood of the mouse are labeled by listing the labels vertically to the right and left of the neighborhood, and drawing lines from each label to its associated feature. More recent work in this vein includes boundary labeling [6].

Our interest is in producing a labeling that at any given point in time looks like a static labeling, but that adapts to a changing viewpoint in a smooth and consistent way, at interactive speeds. The immediate predecessor for our work is Petzold et al. [18, 19, 21, 20], who solve the dynamic map labeling problem in two phases. In the **preprocessing phase**, they compute a data structure called the "reactive conflict graph"; in the **interaction phase**, this reactive graph is queried to obtain a static conflict graph G whose nodes are all the map features within the current view, and whose edges indicate potential conflict for labels of these nodes. From G, they finally select a subset of labels that can be placed without conflict at the current scale. This is done by the greedy method: assuming a priority ordering among labels, the labels are placed (or rejected) in priority order. They do not address what we call the **label consistency** problem.

Poon and Shin [22] consider zooming over a set of point labels—i.e., labels for point features, such as towns and cities. They consider only axis-parallel, rectangular labels, where the point feature must be on the left boundary of the label. As we zoom out, the labels grow to the right and vertically. This is a restricted version of our invariant point placements. The one-dimensional version of this problem can be solved for a given scale with greedy interval scheduling. To allow for zooming they build an $O(\log n)$ height hierarchy. Each level contains the greedy solution on the points in the next lower level, but at a higher scale. To label a window, first find the most appropriate level in the hierarchy, and then pull out the optimal solution at that level. For the two-dimensional problem, they apply the one-dimensional solution to the labels intersecting a set of horizontal stabbing lines. This solution has relatively high time and space requirements for run-time querying—for example, for label filtering they build a range tree at each level of detail.

The labeling problem has some similarity to the settlement selection problem [28, 26], which is to choose the towns and cities that will be shown. Several heuristics for settlement selection are described in [28, 26], but no consideration is given to consistency during zooming.

Work on labeling of 3D illustrations [2] and labeling in virtual and augmented reality systems [7, 3] also relates to our dynamic labeling problem, in that they discuss some heuristics for improving the "frame coherence" or "temporal continuity" of the labeling.

We may note that map labeling is closely connected to labeling of graph drawings [16, 4].

Although the algorithmic literature on label selection as a standalone problem seems to be non-existent, there are several papers discussing heuristics for selecting labels. E.g., Tatemura [24] discusses "dynamic label sampling" in the context of fisheye-view maps. He does not require non-overlap among labels and/or features. The selection is based on considerations such as the avoidance of clutter, distance from area of interest, etc.

3 Framework for Fast and Consistent Labeling

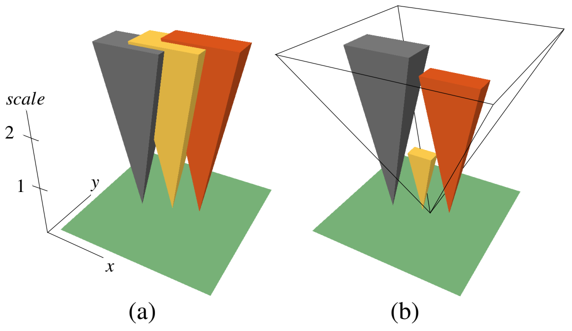
In this section we give an overview of our dynamic labeling approach; in Section 4 we formally define the problem and our solution.

We can think of static map labeling as being composed of two operations, which may be intertwined: **selection** and **placement**. From the set of all possible labels, we need to first select a subset, and then place each label in the subset such that no two labels overlap. Placing a label means determining a size, orientation and location on the map. Now we need to think about what selection and placement mean in a dynamic environment. While the static labeling problem is essentially two-dimensional, the dynamic labeling problem requires a third dimension. It might seem that time would be a natural third dimension, but such a labeling problem would be extremely hard to model in a meaningful way. Instead, we take **scale** as our third dimension. Under this model, the **dynamic selection** problem is to determine at which scales a label will be selected, and the **dynamic placement** problem is to determine a (static) placement for each scale at which a label is selected. Since this approach fixes a single static placement per label per scale, it restricts the scope of our model to labels that don't "slide around" under panning.

Desideratum (D2) says that a label's position and size should vary continuously with the pan and zoom operations, and we have said that in our model the placement must be a function of scale—i.e., it's position is fixed under panning. Therefore, we can visualize each dynamic placement in world coordinates as an extruded label shape, with the vertical dimension being scale. See Figure 3(a). Since we have not yet considered selection, we can imagine these extrusions as being defined for all positive scales. In our implementation we use a simple form of dynamic placements called **invariant point placements**, in which the extrusion is a cone. In this case, the label size in world coordinates is proportional to scale, which means the screen size of the label is invariant under zooming. We say that such placements satisfy the **label size invariance property**. The dynamic placements in Figure 3 are invariant point placements, with rectangular labels.

Desideratum (D1) says that a label should not appear, disappear, and reappear under monotonic zooming. This greatly simplifies the dynamic selection problem. It means that each label must be selected precisely within a single scale interval; we call the selected interval for each label its **active range**: $A^L := [s^L_{\min}, s^L_{\max}]$. Under the strict interpretation of (D1), a label may not disappear when zooming in. In that case we must have $s^L_{\min} = 0$. If we start with the extrusion defined by a label L and its dynamic placement, and restrict that extrusion to L's active range, we are left with a **truncated extrusion**. See Figure 3(b). We can ensure that no two labels will overlap at any scale if we do placement and selection in such a way that no two truncated extrusions overlap. (More precisely, the interiors do not intersect; intersection at boundary points is allowed.)

Figure 3(b) also shows the outline of the view window cone. A horizontal slice of this graph, at a fixed scale, gives the 2D map that



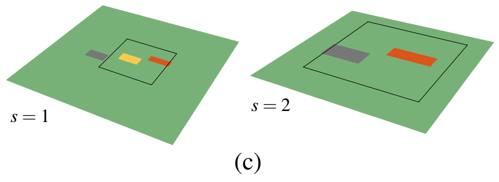


Fig. 3. (a) Dynamic placements for three labels, in world coordinates. These are invariant point placements that satisfy the label size invariance property—screen size is fixed, so size in world coordinates is proportional to scale. (b) The same dynamic placements, but truncated to active ranges so that no two labels intersect at any scale. An outline of the view window cone is also shown. (c) Horizontal slices of the truncated placements at $s = 1$ and $s = 2$. The view area is shown in outline.

is specified by this dynamic labeling. Two such slices are shown in Figure 3(c). At scale $s = 1$ all three labels are active, but only the gold and red ones are inside the view area. At $s = 2$ the view area has expanded (because of zooming out) to encompass all three labels, but the gold label is not active because of the conflict with the red one. As we zoom out from $s = 1$ to $s = 2$ (remember that we assume continuous zooming), the gray label slides into the view area, and all three labels slide toward each other. At some scale between $s = 1$ and $s = 2$ the gold label disappears because of the conflict with the red one. Since these labels have the label size invariance property, they get bigger in world coordinates as we zoom out, but they remain the same size relative to the view area.

For a dynamic map, the need for interactive speed means that we must introduce the **filtering** operation, in addition to selection and placement. The number of labels in the entire map is much larger than can be shown in any given view area, so we can often quickly remove from consideration a large portion of the labels. We can filter on the basis of geographic region, in which case we throw out any label that does not intersect the current view area. We can also filter on the basis of scale—for example, if we are zoomed out far enough, we can throw out all labels for small neighborhood streets.

As noted in Section 1, a natural first attempt at interactive dynamic labeling would be to first filter the labels (on the basis of scale and/or region), and then run a static placement algorithm on that much smaller subset. We believe that such an approach would not be fast enough for interaction. More importantly, it doesn't seem possible to satisfy our consistency desiderata with this approach. (See Figure 1.) Therefore, we propose an inversion of the normal order of doing these operations: place, then select, and finally filter. We achieve interactive speed by moving placement and selection into the preprocessing phase. No label conflict computations are performed in the interaction phase, as it amounts to retrieving a precomputed selection and placement of the labels.

Now we can describe our algorithmic framework:

Preprocessing phase

1. Determine a **dynamic placement** for each label. This is just a static placement at each scale. In this step we consider each label

in isolation, ignoring conflicts with other labels. The dynamic placement should be continuous with scale.

2. Choose an **active range** for each label, such that the resulting truncated extrusions are pairwise non-overlapping.

Interaction phase

1. Filter the labels on the basis of geographic region and/or scale.

2. For each label that has not been filtered out, display it if and only if the current scale is within its active range.

This framework clearly satisfies our consistency desiderata and guarantees that no two labels will ever overlap on screen, yet still allows flexibility in choosing placement and selection algorithms. We only require that placement be a continuous function of scale and selection take the form of active ranges. In fact, there is no a priori reason that placement needs to be done before selection. We have used that order to get a solution that is simple and fast, but in theory selection could be done first or the two could be intertwined.

In the context of this framework, we want selection and placement algorithms that are in some sense optimal. Informally, optimality amounts to showing "as many labels as possible." In a static map it is clear what is meant by "the number of labels selected." For a dynamic map, we can integrate this number over all scales, and with active ranges the integral reduces to a simple sum. Thus, we arrive at the following dynamic labeling optimization problem:

> Given a set of labels S_0, determine a dynamic placement and an active range (i.e., a truncated extrusion) for each $L \in S_0$, such that no two truncated extrusions overlap, and the number of labels active over all scales is maximized. Specifically, maximize $\sum_{L\in S_0} s^L_{\max} - s^L_{\min}$.

Note that at this point we treat each label equally; there is no sense of label priority. Also, for this to make sense we must assume that every active range has a finite upper bound. Let $s_{\max}$ be the highest scale at which any label will ever be shown, so that for every label L, we will enforce $s^L_{\max} \le s_{\max}$. For simplicity, let's also take $s^L_{\min} = 0$. This amounts to the more restricted version of (D1), in which labels do not disappear when zooming in.

This problem appears to be quite difficult. In Appendix A we show that the simpler problem of computing an optimal set of active ranges, *given* a set of dynamic placements, is *NP*-complete for a suitably generalized class of label shapes. To come up with a practical solution, we now introduce label priorities. Priorities are used by [21, 20] during the interaction phase only; in contrast, we shall exploit priority in the preprocessing phase. We assume that each label has a unique priority. We also assume that whenever two labels "fight" over map space, the higher priority label must "win". To make this notion more precise, say that label L is **blocked** by label L' if expanding L's active range by an infinitesimal amount would cause the two truncated extrusions to overlap. In Figure 3(b), the gold label is blocked by the red one, and the red label is blocked by the gray one. Now, the **priority rule** is simply that if L is blocked, it must be blocked by a higher priority label. (It might coincidentally also be blocked by a lower priority label.)

With label priorities, we can find a solution to the following restricted form of the labeling optimization problem:

> Given a set of labels S_0 and a dynamic placement for each $L \in S_0$, choose an active range for each $L \in S_0$ such that $s^L_{\min} = 0$ for every L, no two truncated extrusions overlap, the priority rule is satisfied, and $\sum_{L\in S_0} s^L_{\max}$ is maximized.

The solution is a simple greedy algorithm, G:

> Consider the labels in order of priority, from highest to lowest. For each label L, if there is a scale at which L would be blocked by a higher priority label, then set $s^L_{\max}$ to the minimum such scale; otherwise set $s^L_{\max} = s_{\max}$. In other words, we make each truncated extrusion as high as we can, without overlapping any previously determined (and higher priority) truncated extrusion.

The optimality of this algorithm is proved in Section 4. Our implementation shows that even with these seemingly severe restrictions, we can still produce a dynamic labeling that is useful and fast. In Section 5 we discuss less restricted forms, including the ability to choose from a set of placements for each label.

4 Dynamic Map Labeling Model

In this section we formally define the concepts introduced in Section 3.

We describe label placement with the language of planar affine transformations. We are interested in three types of transformation $\tau : \mathbb{R}^2 \to \mathbb{R}^2$: (i) $\tau = T(e, f)$ is translation by the vector $(e, f) \in \mathbb{R}^2$, (ii) $\tau = R(\theta)$ is rotation by angle $\theta \in [0, 2\pi)$, and (iii) $\tau = D(s)$ is dilation (uniform scaling) by factor $s \in \mathbb{R}_{>0}$. Let us call **allowable** any composition of these 3 types of transformations. Each τ can be represented by a 3×3 matrix M_τ, with composition of transformation corresponding to matrix multiplication. Let $\det(\tau)$ refer to the determinant of M_τ. Clearly $\det(T(e, f)) = \det(R(\theta)) = 1$ and $\det(D(s)) = s^2$.

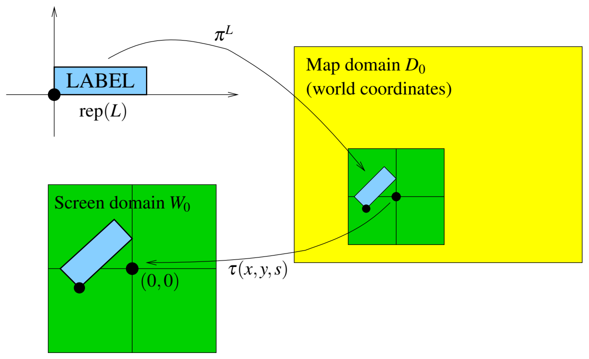


Fig. 4. Transformations among world, screen and label coordinates.

Figure 4 illustrates static placement. Let D_0 denote the domain of map M. D_0 is a rectangular region of $\mathbb{R}^2$, and is defined in **world coordinates**. (In a more realistic model D_0 would be a patch of the surface of a sphere). We view the map M as a collection of **map features**, and there are basically three of interest: point, line and area features. Each feature is viewed as a subset of D_0. Note that lines and areas are really polygonal lines and simply connected polygonal regions.

Let W_0 be the window, in **screen coordinates**. The axis-aligned rectangular subset of D_0 that is displayed in W_0 at a given point in time is called the **view area**. The view area is defined by the width w and height h of W_0 in screen coordinates, a center point $(x, y) \in D_0$, and a scale s. We define s in such a way that a $w \times h$ window corresponds to a $ws \times hs$ view area. Thus, increasing the scale corresponds to zooming out.[1] For simplicity we assume that w and h are fixed, so that a view area is given by $W(x, y, s) \subseteq D_0$, and the current **state** of the view is (x, y, s). At interaction time, the user issues a sequence of commands of the form "Pan to (x, y)" or "Zoom to s".

If we define the window as having center $(0, 0)$ in screen coordinates, then the transformation $\tau(x, y, s)$ that takes $W(x, y, s)$ to W_0 is a translation followed by a dilation:

$$\tau(x, y, s) = D(1/s) \circ T(-x, -y).$$

Note that $\det(\tau(x, y, s)) = 1/s^2$.

We generally think of a label L as a character string, but it could be an icon or a combination of characters and icons. For our purposes, regard L as a compact, simply connected set $\text{rep}(L) \subseteq \mathbb{R}^2$, representing the canonical rendering of L in its own **label coordinate system**. Typically rep(L) is a rectangular box (Figure 4). L is also associated with the map feature $\phi(L) \subseteq D_0$ that it labels.

[1]In cartography, a "large-scale map" refers to one that shows a smaller area in greater detail. This is the inverse of our definition of scale, which may be called **zoom scale**. A zoom scale of s corresponds to a map scale of $1 : s$.

A **static placement** for a label L is an allowable transformation π (or π^L) from L's label coordinates into world coordinates—rep(L) is mapped into some translated, rotated, dilated shape $\pi(\text{rep}(L))$. We say that π is **valid** if it meets certain cartographic constraints, which typically depend on the nature of the feature $\phi(L)$. Intuitively, we expect $\pi(\text{rep}(L))$ to be near to the point or line feature that it labels, and to overlap the area feature that it labels. Such considerations are outside the scope of this paper.

A label L is **visible** under placement π in state (x,y,s) if $\pi(\text{rep}(L)) \cap W(x,y,s)$ is non-empty. In screen coordinates, this placement of L shows up in the region $(\tau(x,y,s) \circ \pi)(\text{rep}(L)) \cap W_0$. See Figure 4.

A **dynamic placement** Π^L for a label L is a function

$$\Pi^L : s \in (0,\infty) \mapsto \pi_s^L$$

that assigns a static placement π_s^L to each scale $s \in (0,\infty)$. Π^L is **valid** if each π_s^L is a valid static placement of L. To meet desideratum (D2), Π^L should vary continuously with s.

Π^L satisfies the **label size invariance property** if there is a constant c such that for every scale s, $\det(\pi_s^L) = cs^2$. By a scaling of L's label coordinates we can, without loss of generality, take $c = 1$. Then $(\tau(x,y,s) \circ \pi_s^L)(\text{rep}(L))$ is a translated, rotated copy of rep(L)—i.e., there is no dilation. Hence rep(L) is already "screen size".

For a set S of labels, a static placement of S is a function

$$\pi^* : L \in S \mapsto \pi^L$$

that assigns a static placement π^L to each label $L \in S$. Similarly, a dynamic placement of S is a function

$$\Pi^* : L \in S \mapsto \Pi^L$$

that assigns a dynamic placement Π^L to each $L \in S$. We say that π^* is **compatible** if for each $L, L' \in S$, $\pi^L(\text{rep}(L))$ and $\pi^{L'}(\text{rep}(L'))$ do not overlap (i.e., their interiors do not intersect).

Let S_0 be the set of all labels. Recall that in our model selection amounts to determining an **active range** $A^L = [s_{\min}^L, s_{\max}^L]$ for each label L. An **active range function** for S_0 is a function

$$A^* : L \in S_0 \mapsto A^L$$

that assigns an active range A^L to each $L \in S_0$.

Say that L is **active** at scale s if $s \in A^L$, and consider the set of labels that are active at scale s: $A^*(S_0;s) := \{L \in S_0 : s \in A^L\}$. We say A^* is **compatible** with Π^* if, for any s, when we use Π^* to place those labels that are active at s under A^*, no two labels will overlap. Formally, let the static placement of $A^*(S_0;s)$ be given by

$$\pi_s^* : L \in A^*(S_0;s) \mapsto \pi_s^L.$$

Then A^* is compatible with Π^* if π_s^* is compatible for every s.

The dynamic labeling optimization problem from Section 3 can now be formulated as follows:

> Given a set of labels S_0, find a dynamic placement Π^* and an active range function A^* such that (i) for each $L \in S_0$, $\Pi^L = \Pi^*(L)$ is valid and varies continuously with scale, (ii) A^* is compatible with Π^*, and (iii) $\sum_{L \in S_0} s_{\max}^L - s_{\min}^L$ is maximized.

Recall that for every label L we will have $s_{\max}^L \leq s_{\max}$, where $s_{\max}^L$ is the upper bound of the interval A^L.

Let E_L refer to the extrusion of rep(L) under a given dynamic placement Π^L, and let E_L/A^L refer to the portion of E_L with $s \in A^L$. We call this a **truncated extrusion**. In the following, we take $A^L = [0, s_{\max}^L]$, as in the restricted form of (D1). A set of truncated extrusions is **compatible** if they are pairwise non-overlapping. This is equivalent to A^* being compatible with Π^*. Say that L is **blocked** by L' under a given Π^* and A^* if increasing $s_{\max}^L$ by an infinitesimal amount would cause E_L/A^L and $E_{L'}/A^{L'}$ to overlap. The **blocking scale** of L and L' under Π^* is the smallest scale s such that E_L/A^L and $E_{L'}/A^{L'}$ intersect if $s_{\max}^L > s$ and $s_{\max}^{L'} > s$. Thus, above scale s, only one of L and L' can be active. (This is called the "cutting scale" in [21, 20].)

Priorities are given by an assignment $P : S_0 \to \mathbb{R}$. Priorities are unique: $L \neq L'$ implies $P(L) \neq P(L')$. We now require that A^* respect the following **priority rule**: If L is blocked, then it is blocked by some L' with $P(L') > P(L)$.

The restricted form of the dynamic labeling optimization problem, from Section 3, becomes

> Given a set of labels S_0 and a dynamic placement Π^* for S_0, choose an active range function A^* compatible with Π^* such that $s_{\min}^L = 0$ for every L, the priority rule is satisfied, and $\sum_{L \in S_0} s_{\max}^L$ is maximized.

Now we can prove the optimality of the greedy algorithm given in Section 3.

LEMMA 1 *Under the priority rule and the restricted (D1), the A^* computed by algorithm G is optimal for any given Π^*.*

Proof. First notice that under any optimal solution every label L must either be blocked or have $s_{\max}^L = s_{\max}$—otherwise we could increase $s_{\max}^L$ to get a better solution. Now suppose there exists an optimal solution $\mathscr{O}$ in which some label has a wider active interval than under G. Let L be the highest priority such label. L must be blocked by some higher priority label L' under G, but not under $\mathscr{O}$. So $E_{L'}/A^{L'}$ is shorter under $\mathscr{O}$ than under G. Since $\mathscr{O}$ is optimal and respects the priority rule, L' must be blocked under $\mathscr{O}$ by some higher priority label L''. But L' is not blocked by L'' under G, which means that $E_{L''}/A^{L''}$ is higher under $\mathscr{O}$ than under G. But $P(L'') > P(L') > P(L)$, which contradicts our choice of L. Therefore, every active interval under G is at least as wide as under any optimal solution, so G is optimal. **Q.E.D.**

The compatibility of A^* with Π^* implies that in the interaction phase, at any scale s at which L is active we can safely display L using π_s^L without worrying about conflicts with other labels. This leads directly to our interaction phase algorithm: to label window $W(x,y,s)$, we first filter the (large) set S_0 to get $S_w \subseteq S_0$, then render each $L \in S_w$ if and only if $s \in A^L$. In contrast to [21, 20], no label conflict computations are done during the interaction phase.

We have omitted one important detail from the selection algorithm, which is how to compute the blocking scale of two labels. This computation is facilitated by restricting ourselves to rectangular labels and invariant point placements. Formally, an **invariant point placement** is a dynamic placement of label L that is represented by (p,q,θ), where p is a point in world coordinates, q is a point in L's label coordinates, and θ is an angle. For every scale $s > 0$ this defines the static placement π_s^L of the form

$$\pi_s^L = T(p)R(\theta)D(s)T(-q)$$

where T, R and D are the translation, rotation and dilation transformations. We see that $\pi_s^L(q) = p$ for all s (this is the "invariant"). For instance, if $q \in \text{rep}(L)$, then at any scale s, $p \in \pi_s^L(\text{rep}(L))$. This is illustrated in Figure 5. Notice also that invariant point placements have the label size invariance property.

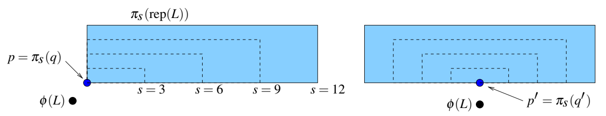


Fig. 5. Two invariant point placements for a point label L, represented by $(p,q,0)$ and $(p',q',0)$ at scales $s = 3,6,9,12$. These are in world coordinates. The feature $\phi(L)$ labeled by L is also shown.

Figure 3 shows invariant point placements on rectangular labels. The regular shape of the extrusions allows us to derive closed form expressions for the location of each label corner as a function of scale, and these can be used to derive the blocking scale for two labels. We omit the details. By computing the blocking scale of all pairs of labels, an $O(n^2)$ implementation of the algorithm to compute A^* from a given Π^* is straightforward. But when we are zoomed in we expect the label conflict graph to be quite sparse, and when we are zoomed out we expect the number of labels active to be quite small, so it seems that we shouldn't need to check every label pair. In Section 5 we describe some heuristics to speed up this computation.

5 Refinements and Practical Considerations

To avoid complicating the presentation, we have delayed discussion of several issues. (i) With a little intertwining of placement and selection, we can intelligently consider multiple possible placements for each label; (ii) we want a way of allowing a liberal version of (D1), in which some labels might disappear when zooming in, but without sacrificing the optimality of the greedy selection algorithm; (iii) we have yet to give a label filtering algorithm; and (iv) we want to improve on the $O(n^2)$ time complexity for computing A^*.

Suppose we have multiple possible dynamic placements for a label, and we don't want to commit to one of them without considering conflicts with nearby labels. For example, we might have four invariant point placements for a point label, corresponding to above left, above right, below left, and below right. A modified greedy algorithm considers all four placements, and chooses the one that allows for the widest active range. Lower priority labels considered later will only be compared with the chosen placement.

Our key to handling the remaining issues is to introduce levels of detail (LOD). Partition the scale interval $[0, s_{\max}]$ into h sub-intervals, each representing a level of detail. (In our implementation, $h = 8$.) Call the LODs $\lambda_i, i = 0, \ldots, h-1$. We have $h+1$ scales, $0 = s_0 < s_1 < \cdots < s_h = s_{\max}$, and each λ_i covers the interval $[s_i, s_{i+1}]$. (For simplicity we'll let the LODs overlap at the boundary scales.)

We associate with each label L a **live range** $R^L = [s_i, s_j]$, with $0 \le i < j \le h$. This indicates that L "wants" to be displayed in levels $\lambda_i, \ldots, \lambda_{j-1}$, absent conflict considerations. For example, a country name might only want to appear in $\lambda_4, \ldots, \lambda_{h-1}$, while a local street name might only appear in λ_0. We require that L may be active only if it is alive: $A^L \subseteq R^L$. Now the liberal version of (D1) is that a label can vanish when zooming in, but only upon reaching the lower end point of its live range.

In our implementation each LOD is labeled separately—each starts with its own set of labels S_0, consisting of those labels that are alive in this level, and placement and selection are computed on this set. This way, when we compute Π^* and A^* for λ_i, we can simply take the lower bound of each active range to be 0, and use the greedy algorithm G as described in Section 3. If a label's live range does not include λ_{i-1}, its disappearing when zooming in beyond s_i is accomplished by simply not including the label in S_0 when placement and selection are done on λ_{i-1}.

The LODs are also treated separately during the interaction phase, so that labeling view area $W(x,y,s)$ only requires searching the LOD that contains s. Thus, filtering on the basis of scale is essentially taken care of in the preprocessing phase. We need to be careful, though, to coordinate between the LODs, so that there is no popping across LOD boundaries. This can be done by computing the labeling in reverse order, from λ_{h-1} down to λ_0. Any label that is active at s_{i+1} in λ_{i+1} should be given a high enough priority in λ_i so that it will also be active at s_{i+1} in λ_i. (We have not yet implemented inter-LOD coordination.)

Filtering on the basis of geographic region must still be done in the interaction phase. For each λ_i we split the map domain D_0 into a grid of buckets, and assign each label to the bucket that contains its invariant point. The labels in each bucket are stored in a list, sorted by $s^L_{\max}$, and the lists are stored in a hash table, indexed by grid location. The filtering step now becomes quite simple: given the boundary of the current view area $W(x,y,s)$, we can easily compute which buckets cover it, and pull those lists out of the hash table. The grid size of a bucket for λ_i is $l \times l$, where l is assumed to be greater than the maximum length of any label at scale s_{i+1}. Then, since a label might span a bucket boundary, we also consider a one bucket wide buffer around $W(x,y,s)$, to be sure that all labels intersecting $W(x,y,s)$ are found.

There is an added benefit to this bucketing approach: since the bucket lists are stored in sorted order, we don't need to traverse an entire list. We only need to step through each list in order of descending $s^L_{\max}$, and stop when we get to a label for which the current scale $s > s^L_{\max}$. Thus, we traverse only the active labels in each list.

Buckets also help to reduce the $O(n^2)$ time for computing A^*. In practice, there really isn't a problem at high LODs (high scales) because so many labels are filtered out by scale. At lower LODs the conflict graph is sparse, so we shouldn't need to check each label against all others. This is where buckets help: a label only needs to be checked for conflicts with other labels in its bucket or in neighboring buckets. In our implementation we have about 12 million labels and half a million buckets in λ_0. So an average of about 24 labels per bucket, but with great variability—from 0 in many sparsely populated areas to several hundred or more in dense urban areas.

6 Conclusions and Future Work

This paper makes theoretical, algorithmic and practical contributions to the area of dynamic labeling. We have formalized the dynamic labeling problem, and given a set of desiderata (D1-D4) for label consistency. Our solution is the first to achieve these desiderata. Prior to our work, there have been no formal models of dynamic maps or dynamic labels. We believe this is an essential step for proving correctness and for elaborating on and comparing various solutions in this domain.

We have formulated a new (and somewhat unintuitive) algorithmic framework for fast and consistent labeling. In this framework, we perform placement and selection of labels in the preprocessing phase, and only filtering during interaction.

We introduced invariant point placements, and a solution for placement and selection based on them. Our solution is highly efficient in the interaction phase, and yet achieves label consistency (D1-D4). Finally, we have validated the usefulness of our approach by a full-scale web demo for a very large data set.

Future work. Many extensions are possible. Within our framework, other possibilities for label placement and selection can be explored. Our invariant point placements are just one reasonably good labeling, but other placements might give more flexibility. We leave open interesting theoretical questions about the complexity of dynamic labeling within our model. We plan to pose dynamic map labeling as an **online problem**, and to introduce competitive analysis for labeling. Finally, we would like to introduce various categories of labels which can be selected or deselected (dynamically) from any view. Besides the standard geographic labels, we can have label categories such as: restaurant, landmark, gas station, public transportation, store, etc. Such extensions are needed to support highly personalized navigation goals in applications.

A Appendix: Complexity of Optimal Active Range Selection

In this section we show that our restricted dynamic labeling optimization problem is *NP*-complete, under the assumption of star-shaped labels. (The weaker result for arbitrary polygonal labels follows immediately.) We do this by reducing the independent set problem for planar graphs to it. The independent set problem is this: given a graph G and an integer k, determine if G has an independent set of size k. The reduction associates the vertices of G with invariant point placements of star-shaped labels, and the edges of G with conflicts between placements. Recall that, as in Figure 3, the extrusions for invariant point placements are cones.

We consider cones in $\mathbb{R}^3$. The last coordinate of points $q \in \mathbb{R}^3$ is called the **height** of q. Given $p \in \mathbb{R}^2$ and $s \in \mathbb{R}$, let $p^{(s)} \in \mathbb{R}^3$ be the point whose height coordinate is s and whose first two coordinates are given by p. If $S \subseteq \mathbb{R}^2$, let $S^{(s)} = \{p^{(s)} : p \in S\}$. A **cone** $C(p,B,h) \subseteq \mathbb{R}^3$ is parametrized by a point $p \in \mathbb{R}^2$; a compact, simply-connected set

$B \subseteq \mathbb{R}^2$; and a height $h \in \mathbb{R}$. The cone comprises all points $q \in \mathbb{R}^3$ of height at most h such that the line $\overline{p^{(0)}, q}$ intersects $B^{(1)}$. Call $p^{(0)}$ the **apex** and h the **height** of the cone. Let $H(C)$ denote the height of cone C. We further assume that B is a star-shaped polygon—that is, a polygon whose interior contains a point that can see all the other points of the interior.

If I is an interval, and C is a cone, let C/I denote the **truncated cone** obtained by restricting the points of C to have height lying in I. If $I \subseteq [0, H(C)]$, we say that I is **compatible** with C. Let $\mathscr{C} = \{C_i : i = 1, \ldots, n\}$ be a set of cones in $\mathbb{R}^3$, and let A be a function that assigns an interval $I_i = A(C_i)$ to each $C_i \in \mathscr{C}$. We say that A is **compatible** if each I_i is compatible with C_i, and the set

$$\{C_i/I_i : i = 1, \ldots, n\}$$

of truncated cones are pairwise non-overlapping. Here, two point sets are said to be overlapping if their interiors intersect. The **active range optimization problem** (ARO) is that of computing a compatible A such that $v(A) := \sum_{i=1}^{n} |I_i|$ is maximized. The **active range decision problem** (ARD) asks, for a given rational number k, whether there is a compatible A such that $v(A) \geq k$. We may simplify the problem by restricting the intervals $A(C_i)$ to the form $I_i = [0, h_i]$, for some $h_i \geq 0$. We then view A as a function that assigns a height $A(C_i) = h_i$ to each cone C_i. Such solutions correspond to the property that labels never disappear while zooming in. For reference, call this the **simple active range decision problem**. For complexity purposes, we will consider the version where the input cones $C = (p, B, h)$ have rational coordinates and where B is a star-shaped polygon; this is the case of *star-shaped map labels*.

THEOREM 2 *The simple ARD problem is NP-complete for star-shaped map labels.*

Proof. It is easy to see that the problem is in *NP*. To show *NP*-hardness, we exploit two facts: the independent set problem for planar graphs is *NP*-hard, and every planar graph $G = (V, E)$ has a straight line embedding (see [23]). To reduce the independent set problem in planar graphs to simple ARD, we will construct a set $\mathscr{C}$ of cones where each cone corresponds to a vertex of the input graph G. Each cone has height $1 + \varepsilon$ (for some $\varepsilon > 0$), and $\mathscr{C}$ has the property that two cones overlap iff there is an edge between the corresponding vertices of G. Moreover, if two cones overlap, then they first touch each other at height exactly 1. Consider a compatible assignment A for these cones: from the said properties, we may assume WLOG that A assigns heights of either 1 or $1 + \varepsilon$ to each cone. Clearly, the set of cones that is assigned height $1 + \varepsilon$ corresponds to an independent set of G. Conversely, for every independent set S, we have a compatible assignment A_S that assigns height $1 + \varepsilon$ to exactly those cones in S. Thus, there is an assignment A such that $v(A) = |V| + t\varepsilon$ iff G has an independent set of size t. Thus, the construction $(G, t) \mapsto (\mathscr{C}, |V| + t\varepsilon)$ is a reduction of the independent set problem in planar graphs to simple ARD. If this reduction can be computed in polynomial time, our *NP*-hardness claim follows. This is given in the next lemma. **Q.E.D.**

The proof of the following lemma makes use of triangular graphs. A **triangular graph** is a maximal planar graph with at least three vertices. Every face of a straight line, planar embedding of a triangular graph is triangular, including the exterior face. Say that a vertex of a triangular graph is exterior if its embedding borders the exterior face.

LEMMA 3 *For any planar graph $G = (V, E)$ and integer $t > 0$, we can compute in polynomial time a set $\mathscr{C}$ of cones for star-shaped polygonal labels, and an $\varepsilon > 0$, such that G has an independent set of size t iff there is a compatible assignment A with $v(A) = |V| + t\varepsilon$.*

Proof. First, we transform the planar graph G to a triangular graph $G' = (V, E')$ by adding edges. By Schnyder's theorem [23], there is a linear time algorithm to compute a straight line embedding of G' in the $(n-2) \times (n-2)$ integer grid, where $n = |V|$. Let $p(u)$ be the embedding for vertex u. Next, for each edge $(u, v) \in E'$, define the point

$$m(u,v) = \begin{cases} \frac{p(u)+p(v)}{2} & \text{if } (u,v) \in E \\ \frac{2p(u)+p(v)}{3} & \text{if } (u,v) \notin E. \end{cases}$$

For each vertex u, form a star-shaped polygon $B(u)$ whose vertices are $m(u, v)$ for each $(u, v) \in E'$. If u is an exterior vertex, we add an additional artificial point in the exterior face of the embedding. Each $B(u)$ so defined is a star-shaped region with $p(u)$ as a center of the star. It is clear that $B(u)$ and $B(v)$ touch iff $(u, v) \in E$.

We now construct a set $\mathscr{C}$ of cones as follows: each $v \in V$ gives rise to a cone $C(p(v), B(v), 1 + \varepsilon)$. The $\varepsilon > 0$ is chosen small enough so that cones $C(p(v), B(v), 1 + \varepsilon)$ and $C(p(u), B(u), 1 + \varepsilon)$ overlap iff $(u, v) \in E$. Now it is easy to see that the original graph $G = (V, E)$ has an independent set of size t iff $\mathscr{C}$ has a compatible solution A with $v(A) = n + t\varepsilon$. **Q.E.D.**

We conjecture that the simple ARD problem is also *NP*-complete for convex polygonal labels, and possibly for rectangular labels as well. We can reduce the independent set problem to simple ARD for circular labels by using Koebe's theorem (1936), which says that every planar graph can be realized as the contact graph of a set of non-overlapping discs. However, it is not known whether these discs can be constructed in polynomial-time. If they can, then the result for convex polygons follows readily, by taking each disc contact point as a vertex of the polygon.

It is not even clear that the 1-dimensional version of the simple ARD problem is polynomial-time.

ACKNOWLEDGEMENTS

The first author would like to acknowledge the support of the Yeshiva University Office of Academic Affairs. The third author was supported in part by NSF Grant CCF-043836 and NSF/ITR Grant CCR-0082056. The authors also thank the reviewers for their constructive comments.

REFERENCES

[1] J. Ahn and H. Freeman. A program for automatic name placement. In *Proc. Symposium on Automated Cartography (Auto-Carto)*, volume 6, pages 444–453, 1983.

[2] K. Ali, K. Hartmann, and T. Strothotte. Label layout for interactive 3d illustrations. *The Journal of WSCG*, 13, 2005.

[3] R. Azuma and C. Furmanski. Evaluating label placement for augmented reality view management. In *Proceedings of IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR 2003)*, pages 66–75, 2003.

[4] G. D. Battista, P. Eades, R. Tamassia, and I. Tollis. Algorithms for drawing graphs: an annotated bibliography, 1994.

[5] K. Been. *Responsive Thinwire Visualization of Large Geographic Datasets*. Ph.d. thesis, New York University, Department of Computer Science, Courant Institute, Sept. 2002. Download from `http://cs.nyu.edu/visual/home/pub/`.

[6] M. A. Bekos, M. Kaufmann, A. Symvonis, and A. Wolff. Boundary labeling: Models and efficient algorithms for rectangular maps. In *Proc. 12th Int. Symposium on Graph Drawing (GD '04)*, 2004. In Lecture Notes in Computer Science, pp. 49–59, 2005.

[7] B. Bell, S. Feiner, and T. Höllerer. View management for virtual and augmented reality. In *ACM Symp. on User Interface Software and Technology (UIST 2001)*, pages 101–110, 2001.

[8] J. Christensen, J. Marks, and S. Shieber. An empirical study of algorithms for point-feature label placement. *ACM Trans. on Graphics*, 14(3):203–232, 1995.

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

[10] J. Fekete and C. Plaisant. Excentric labeling: Dynamic neighborhood labeling for data visualization. In *Proc. ACM CHI'99*, pages 512–519, 1999.

[11] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *ACM Symp. on Comp. Geometry*, volume 7, pages 281–288, 1991.

[12] H. Freeman. AUTONAP – an expert system for automatic map name placement. In *Proc. 1st International Symp. on Spatial Data Handling*, pages 544–569, 1984. Universität Zürich-Irchel.

[13] G-Vis Dynamic Labeling Demo, 2002. URL `http://sage.mc.yu.edu/gvis/`.

[14] S. A. Hirsch. An algorithm for automatic name placement around point data. *The American Cartographer*, 9(1):5–17, 1982.

[15] E. Imhoff. Positioning names on maps. *The American Cartographer*, 2(2):128–144, 1975.

[16] K. G. Kakoulis and I. G. Tollis. A unified approach to automatic label placement. *Int'l. J. Comput. Geometry and Appl.*, 13(1):23–60, 2003.

[17] J. Marks and S. Shieber. The computational complexity of cartographic label placement. Technical Report TR-05-91, Harvard CS, 1991.

[18] I. Petzold. Textplazierung in dynamisch erzeugten Karten. Diploma-thesis, Institute for Computer Science, Bonn, 1996.

[19] I. Petzold. *Beschriftung von Bildschirmkarten in Echtzeit – Konzept und Struktur*. Ph.d. thesis, Institute of Cartographcy and Geoinformation, University of Bonn, 2003.

[20] I. Petzold, G. Gröger, and L. Plümer. Fast screen map labeling – data-structures and algorithms. In *Proc. 21th International Cartographic Conferences (ICC'03)*, pages 288–298, 2003.

[21] I. Petzold, L. Plümer, and M. Heber. Label placement for dynamically generated screen maps. In *Proc. 19th International Cartographic Conferences (ICC'99)*, pages 893–903, 1999. Ottawa, Canada.

[22] S.-H. Poon and C.-S. Shin. Adaptive zooming in point set labeling. In *Proceedings 15th International Symposium on Fundamentals of Computation Theory (FCT 2005)*, pages 233–244, 2005.

[23] W. Schnyder. Embedding planar graphs on the grid. In *Proc. 1st ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 138–148, 1990.

[24] J. Tatemura. Dynamic label sampling on fisheye maps for information exploration. In *Advanced Visual Interfaces 2000*, pages 238–241, 2000.

[25] Topologically integrated geographic encoding and referencing system (tiger) homepage, 1995. US Census Bureau. URL http://www.census.gov/ftp/pub/geo/www/tiger/.

[26] M. van Kreveld, R. van Oostrum, and J. Snoeyink. Efficient settlement selection for interactive display. In *Proceedings AutoCarto 13: ACSM/ASPRS '97 Technical Papers*, pages 287–296, 1997.

[27] M. J. van Kreveld, J. Nievergelt, T. Roos, and P. Widmayer, editors. *Algorithmic Foundations of Geographical Information Systems*, volume 1340. Springer, 1997. CISM Advanced School on the Algorithmic Foundations of Geographical Information Systems, Udine, Italy, Sep 16-20, 1996.

[28] R. W. van Oostrum. *Geometric Algorithms for Geographic Information Systems*. PhD thesis, Universiteit Utrecht, 1999.

[29] F. Wagner and A. Wolff. A practical map labeling algorithm. *Comput. Geometry: Theory and Appl.*, 7:387–404, 1997.

[30] A. Wolff. The map-labeling bibliography, 2005. http://illwww.ira.uka.de/˜awolff/map-labeling/bibliography/.

[31] C. Yap, K. Been, and Z. Du. Responsive thinwire visualization: Application to large geographic datasets. In E. et al., editor, *Proc. SPIE Symp. on Visualization and Data Analysis 2002*, volume 4665, pages 1–12, 2002. 19-25 Jan, 2002, San Jose, California.

[32] P. Yoeli. The logic of automated map lettering. *The Cartographic Journal*, 9(2):99–108, 1972.