Point Set Labeling with Sliding Labels*

Marc van Kreveld
Dept. of Computer Science
Utrecht University
marc@cs.ruu.nl

Tycho Strijk
Dept. of Computer Science
Utrecht University
tycho@cs.ruu.nl

Alexander Wolff
Dept. of Computer Science
Free University Berlin
awolff@inf.fu-berlin.de

Abstract

This paper discusses algorithms for labeling sets of points in the plane, where labels are not restricted to some finite number of positions. We show that continuously sliding labels allows more points to be labeled both in theory and in practice. We define six different models of labeling, and analyze how much better—more points get a label—one model can be than another. Maximizing the number of labeled points is NP-hard, but we show that all models have a polynomial-time approximation scheme, and all models have a simple and efficient factor-$\frac{1}{2}$ approximation algorithm. Finally, we give experimental results based on the factor-$\frac{1}{2}$ approximation algorithm to compare the models in practice.

Figure 1: Rectangular labels of cities of the U.S.A.

1 Introduction

Annotating sets of points is a common task to be performed in Geographic Information Systems. Cities on small-scale maps are shown as points with the city's name attached (Figure 1 shows names as rectangles), points of altitude usually are small "+"-signs with a value, and in point pattern analysis [2], points in a plot are labeled with a sequence number. In (spatial) statistics [15], point sets are also common in data postings of field measurements, scatterplots of principal component analysis, and variograms, for instance.

Generally it is assumed that a point label can be seen as an axis-parallel rectangle, the bounding box of the text (see Figure 1). Several algorithms for point feature labeling have been described in the automated cartography literature and in computational geometry (far too many to list; for bibliographies see [8, 21]). The more general problem of map labeling includes line feature labeling (roads, rivers) and area labeling (countries) as well.

A good label placement for the points of a given set has two basic requirements. A label should be placed with the point to which it belongs, and two labels should not overlap each other. For high quality cartographic label placement, more requirements can be formulated such as unambiguity [14, 22]. Given these basic requirements, an algorithm can try to either label as many points as possible, or find the largest possible font such that all points can be labeled. Labeling as many points as possible already give rise to a computationally intractable problem under the two basic conditions [9, 10, 16].

Nearly all of the existing algorithms for point annotation limit the placement of a label with respect to its point to a finite number of possible positions. Algorithms described before usually allow four label positions (the point is at one of the four corners) [9, 19, 20], eight (many papers in the automated cartography literature), or any constant number [1, 7]. We call restrictions of the allowed label positions the *model* that is used by the algorithm. Models that allow a finite number of positions per label are *fixed-position models*.

In this paper we drop the restriction that a label can only be placed at a finite number of positions. Instead, we allow any position on the edges of the rectangle to coincide with the point, see Figure 1. Such a model is called a *slider model*. We will study how many more labels can be placed with slider models than with fixed-position models, and to what extent slider models require more difficult algorithms. We generally assume

*This research is supported by the Dutch Organization for Scientific Research (N.W.O.), the ESPRIT IV LTR Project No. 21957 (CGAL), and by the Deutsche Forschungsgemeinschaft (DFG) under grant Wa 1066/3-1.

that labels have equal height but not necessarily the same width. This is a natural assumption if labels contain text or numbers of a fixed font size. We consider the rectangle that represents a label to be closed, which implies that labels are not allowed to touch.

Hirsch [11] has also used a slider model for labeling points. However, his approach is completely different from ours. He defines repelling forces for overlapping labels and computes translation vectors for them. After translation, this process is repeated and hopefully, a labeling with few overlaps appears after a number of iterations.

This paper is structured as follows. Section 2 introduces the six models—three fixed-position and three slider—that are compared in this paper. We analyze how many more labels can be placed in one model than another, in theory. In Section 3 we show that the slider models allow a simple factor-$\frac{1}{2}$ approximation algorithm that uses $O(n)$ space and $O(n \log n)$ time. This was already known for the fixed-position models. In Section 4 a polynomial time approximation scheme is given, showing that for any constant $\epsilon > 0$, a polynomial time algorithm exists that labels a fraction of at least $1-\epsilon$ of the optimal number of labels that can be placed. Again, this result was already known for the fixed-position models but not for the slider models. We remark that our algorithms can be adapted for labels of varying height, but the approximation factors don't hold any more.

Section 5 contains a comparison of the six different models in practice. For these models, the factor-$\frac{1}{2}$ approximation algorithms have been implemented and tested on real world data. We used three different data sets. One contains 1000 cities of the U.S.A., another contains a data posting with 236 measurements, and the third contains 75 points in a scatterplot near a regression line. Here the labels are the sequence numbers of the points. We give tables showing how many points are labeled in each model. It appears that the slider models produce about 10–15% more labels than the fixed-position models. This improvement is significant, because more labels are placed in the difficult areas.

2 Comparing label models

This paper considers three fixed-position models and three slider models for point feature label placement. These are shown in Figure 2. It is obvious that a maximum labeling of a set P of points in the 4-slider model must have as least as many labels as in any of the other models. Similarly, the 1-slider model is at least as good as the 2-position model. The partial order representing the generality of the models, shown in Figure 3, presents itself immediately. It is clear that for some point sets all points can be labeled in any of the models. But the question "How many more points can be labeled in one model than another, for *some* point set?" is more difficult to answer. This section deals with this question. For simplicity, all labels in this section are squares of unit size.

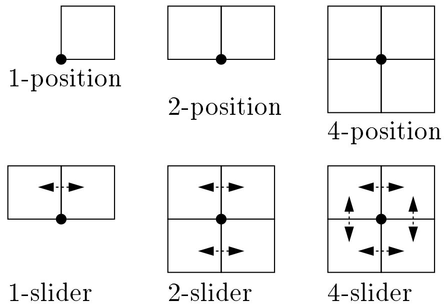


Figure 2: Fixed-position and slider models.

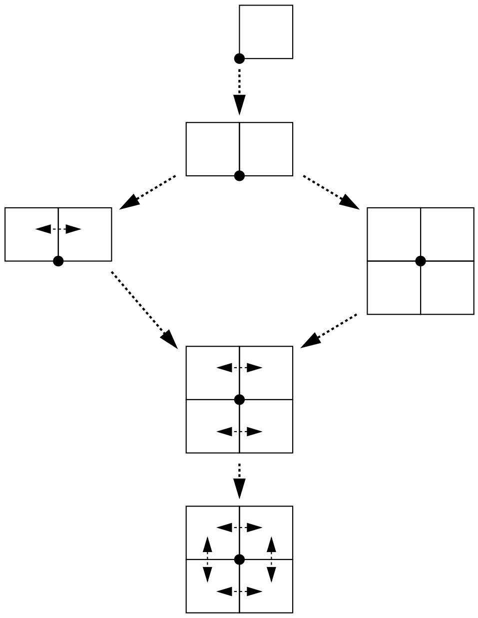


Figure 3: Partial order on the models.

Let P be a set of n points in the plane. Let M_1 and M_2 be two models for labeling P, and let $\mathrm{opt}_{M_1}(P)$ and $\mathrm{opt}_{M_2}(P)$ be the maximum number of points of P that receive a label in the models M_1 and M_2, respectively. The *ratio of the models* M_1 *and* M_2 is the supremum of the ratio $\mathrm{opt}_{M_1}(P)/\mathrm{opt}_{M_2}(P)$ for $n \to \infty$ and maximized over all point sets P with n points. Figure 4 shows, for instance, that the 2-position model can be twice as good as the 1-position model, which is not really surprising. We can also show that the ratio of 2 is tight. Consider an maximum labeling of k points from a point set P with n points, using the 2-position model. Move the labels of these k points into the position al-

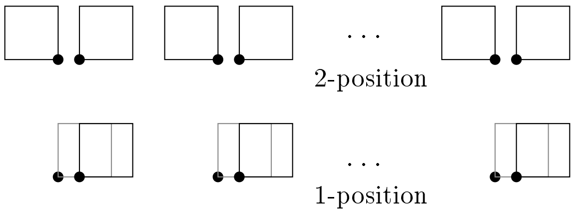


Figure 4: Optimal labeling of P in two models.

lowed in the 1-position model. Either $\lceil k/2 \rceil$ of the labels were already in this position, and they form a nonintersecting set, or $\lceil k/2 \rceil$ of the labels were in the other position and have been *flipped*. Since these squares were nonintersecting before flipping, they must also be nonintersecting after flipping, which proves that the ratio of 2 is tight. The same simple idea gives a ratio of 2 between the 1-slider and 2-slider models, and the 2-position and 4-position models. To analyze the ratios between other models more complicated variations of this idea are needed.

To analyze the ratios between models M_1 and M_2, we use the following strategy. We want to bound the ratio ρ by which more labels can be placed in the model with more freedom, say M_1. We assume an optimal label placement in M_1. Then we canonically re-label the labeled points by moving every label into a position which is valid in the more restrictive model M_2. This may cause some labels to intersect. We determine the maximum number δ_{left} of M_2-labels that intersect the leftmost M_2-label l. Then we put l into a set S, remove l and all its conflict partners from the instance and repeat until no labels remain. At the end of the process, S contains at least $\text{opt}_{M_1}(P)/(\delta_{\text{left}}+1)$ non-intersecting M_2-labels, where $\text{opt}_{M_1}(P)$ is the size of the assumed optimal M_1-placement. The size of S is a lower bound for the size of an optimal M_2-placement, thus $\delta_{\text{left}}+1$ is an upper bound for the M_1, M_2-ratio.

To prove a lower bound for the ratio of two models, we need to give an example of arbitrary size for which any M_2-placement is worse by this ratio.

Lemma 1 *Given two labeling models M_1 and M_2 such that the labels in M_1 can be slid into some position of M_2 using only horizontal sliding or only vertical sliding (not both). Let ρ be the M_1, M_2-ratio. Then $\rho \leq 3$.*

Proof: Again we consider an optimal M_1-labeling of an arbitrary instance. Assume that we can slide M_1- into M_2-label positions vertically (for instance, M_1 is the 4-slider model and M_2 is the 2-slider model). We canonically slide all M_1-labels that are not yet in an M_2-label position upwards, until they arrive in an M_2-label position. We show that the leftmost M_2-label l_2 can then intersect at most two other M_2-labels. This yields the upper bound of 3 for ρ.

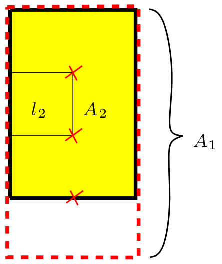


Figure 5: If M_1 can be slid into M_2 then the leftmost M_2-label l_2 cannot intersect more than two M_2-labels.

M_2-labels intersecting l_2 can only lie within area A_2 in Figure 5 since l_2 is leftmost. The corresponding M_1-labels are restricted to area A_1. Every label in A_1 must contain one of the three gridpoints in the interior of A_1 (marked by crosses in Figure 5). Thus A_1 can contain only three non-intersecting M_1-labels including the M_1-counterpart of l_2. It follows that l_2 cannot intersect more than two M_2-labels, and that $\rho \leq 3$. ∎

A simple example gives a lower bound of 2 for the ratio of, for instance, the 2-slider model and the 4-slider model. Consider the set of n points $\{(0, i) \mid 0 \leq i \leq n-1\}$. Then the 2-slider model can label at best every other point, giving $\lceil n/2 \rceil$ labels. The 4-slider model can label the point at the origin right and below, and every next point with a label shifted $1 + 1/(n-1)$ upwards with respect to the label below. This example makes explicitly use of the assumption that squares be closed, but this is not necessary: use two columns of points at x-distance $\frac{1}{2}$, and use $1-\epsilon$ as y-distance between the points.

In the full paper, we show the upper and lower bounds on the ratios between two models given in Figure 6. The lower bound $2\frac{1}{4}$ appears from an example where the 1-position model gives a 9-cycle of intersecting squares, but the 1-slider model allows a non-intersecting placement.

Finally, we wish to note that slider models can be 3/2 times as good as *any finite approximation* of that slider model. For example, consider the 1-slider model and a model that allows a fixed set of one hundred different positions where the bottom edge of the square coincides with the point to be labeled. Then we take three points $(0, 0)$, $(x, 0)$, $(1+\epsilon, 0)$, choose $\epsilon > 0$ to be very small, and choose x such that none of the one hundred allowed positions fits between the outer two points. This is always possible. The 1-slider model can label all three points regardless of the choice of ϵ and x. By copying these three points $n/3$ times we get the ratio 3/2.

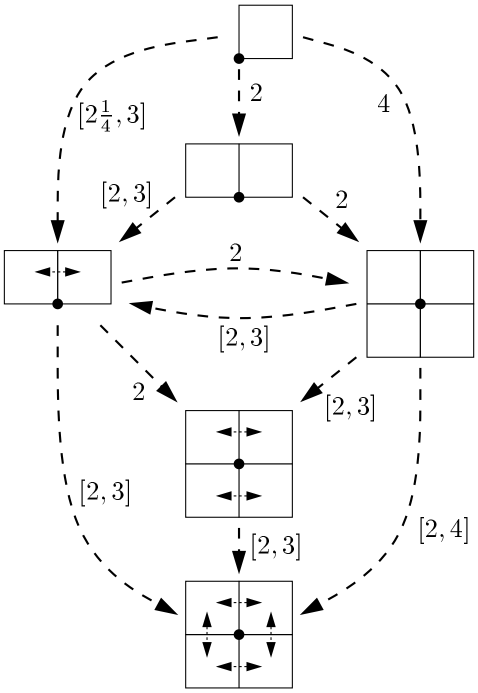


Figure 6: Upper and lower bounds for the ratios of two models.

3 Greedy approximation algorithms

In the following sections we consider algorithms for point feature labeling in the slider models. Labeling the maximum number of points is intractable in the fixed-position models [9, 10, 16], so one has to be content with approximation algorithms. In this section we describe a simple $O(n \log n)$ time algorithm for the slider models that approximates an optimal solution in the following sense. If the maximum number of labels that can be placed is K, then our algorithm places at least $K/2$ labels: a factor-$\frac{1}{2}$ approximation algorithm. In most data sets, however, we expect to come much closer to the optimum.

For the fixed position models, simple $O(n \log n)$ time, factor-$\frac{1}{2}$ approximation algorithms were described recently by Agarwal et al. [1]. We obtain the same result for the slider models. We'll only describe the most general 4-slider algorithm; it is an extension of the 1-slider and 2-slider algorithms. It is based on a greedy strategy. For convenience we'll do as if labels were allowed to touch, unlike in the previous section. In the full paper we show that simple adaptations can be made to obtain non-touching labels.

Given a set of points with labels that have already been placed, and a set of points that don't have a label yet, define the *leftmost label* to be the label whose right edge is leftmost among all possible labels of unlabeled points. So by definition, the leftmost label doesn't intersect any label that has been placed. The strategy is the same as for fixed position labels [1] but the algorithm is quite different. The proof of the following lemma is given in the full paper.

Lemma 2 *Given labels of fixed height and any of the slider models, the greedy strategy of repeatedly choosing the leftmost label finds a labeling of at least half the number of points labeled in an optimal solution.*

Let $\{p_1, \ldots, p_n\}$ be the set of points that has to be labeled. The label of p_i is denoted l_i, and the reference point of a label is its lower left vertex. The possible positions of the reference point of a point p_i are represented by four line segments. Two are horizontal, h_{2i-1} and h_{2i}, and two are vertical, v_{2i-1} and v_{2i}. Their position is exactly the position of the edges of the label l_i if it were placed left and below p_i. The width of l_i is denoted w_i, and the height is always 1 (we can normalize to this situation).

If a label l_i has been placed, then no reference point position inside l_i is possible. The same holds for reference points inside the rectangle l'_i precisely one unit below l_i (since any label extends one unit above its reference point). Furthermore, since labels are placed from left to right, no reference point positions to the left of l_i and l'_i will still be accepted by the algorithm. Suppose a subset of the points has already received labels by the algorithm. The *right envelope* of all labels l and their

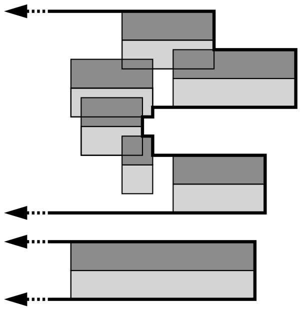


Figure 7: Frontier of the placed labels (dark grey) and their lowered copies (light grey).

copies l' outlines all reference point positions that are impossible, or cannot occur any more, see Figure 7. We call this right envelope the *frontier* and denote it by F.

To determine the next leftmost label, we only have to consider the frontier F and the segments $h_{2i-1}, h_{2i}, v_{2i-1}$, and v_{2i} of the points p_i to the right of F that don't have a label yet. Given a horizontal segment h and the frontier F, there are three possibilities: (i) h lies completely left of F. Then h can be discarded; a point on it cannot be a reference point for

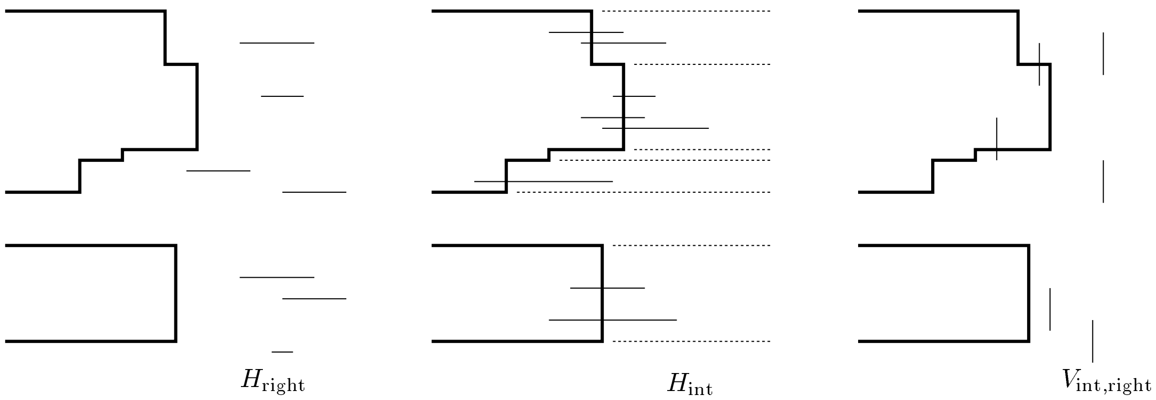


Figure 8: The sets H_{right}, H_{int}, and $V_{\text{int,right}}$. The dashed lines in the middle picture separate the segments of H_{int} that are in different red-black trees $\mathcal{T}_i$.

a label that doesn't overlap another label. (ii) h lies completely right of F. Then the leftmost point on h is a candidate for the next leftmost label. (iii) h intersects F. Then a point just right of the intersection point is the candidate. For a vertical segment v, a similar situation occurs. If v lies left of F, it can be discarded; if v lies right of F, any point on v can be chosen; and if v and F intersect, then any point on v right of F can be chosen as a candidate.

Let H be the set of all horizontal segments that represent reference points of the labels. Similarly, let V be the set of the corresponding vertical segments. Let $H_{\text{right}} \subseteq H$ be the subset of all horizontal segments that lie completely right of F. See Figure 8. Let $H_{\text{int}} \subseteq H$ be the subset of all horizontal segments that intersect F. Let $H_{\text{left}} \subseteq H$ be the subset of all horizontal segments that lie completely left of F (these cannot give a valid label any more). Let $V_{\text{int,right}} \subseteq V$ be the subset of all vertical segments that contain at least some point right of F.

To maintain the frontier and the candidates for the best reference point we'll use a few data structures. Some of the data structures are used to find the next leftmost label; other data structures are only used to update the former ones efficiently. The data structures are red-black trees $\mathcal{T}$, heaps $\mathcal{H}$, and priority search trees $\mathcal{P}$ [17]. These are also described in standard textbooks on algorithms [5] and computational geometry [6]. We use three data structures to find the leftmost label position among the ones represented by H_{int}, H_{right}, and $V_{\text{int,right}}$.

Leftmost label query structures.

1. For each segment in H_{right} we store the x-coordinate of its right endpoint. This corresponds to the right edge of a label whose reference point is the left endpoint of the segment. These values are stored in a heap, where the root stores the minimum.

2. The subset H_{int} is stored as follows. For each vertical segment f_i of F, we maintain a red-black tree $\mathcal{T}_i$ with the segments in H_{int} that intersect f_i (see the middle picture of Figure 8). These are stored in the leaves sorted on y-coordinate. With each leaf we also store the width of the corresponding label. We augment each red-black tree by storing at each internal node the minimum width label in the subtree of that node [5]. We also use a heap $\mathcal{H}_{\text{int}}$. It stores for each $\mathcal{T}_i$ the x-coordinate of f_i plus the minimum width. The root of the heap stores the minimum of these sums, which corresponds to the leftmost label among those represented by H_{int}.

3. For the vertical segments in V, we don't store $V_{\text{int,right}}$ but some set V' for which $V_{\text{int,right}} \subseteq V' \subseteq V$. The x-coordinate of each segment of V' is stored in a heap. The heap may return as the minimum some segment that lies completely left of F, so it may also contain labels that cannot be placed. After extracting the minimum from the heap, we test if it is in $V_{\text{int,right}}$. If not, we discard it and extract the next minimum from the heap, until we find one in $V_{\text{int,right}}$.

We query the three heaps described above. Among their answers, one corresponds to the leftmost label. This is the label we place. Then we must update the frontier F and several of the data structures described above. This is not so easy. We'll use some more data structures that help to do the updating after the frontier has changed. Note that the frontier can only change in one way. When a new label is placed, the new frontier is the right envelope of the old frontier, the right edge of the newly placed label l, and the right edge of the downward shifted copy l' of l. This vertical edge is denoted f_{new} and has twice the unit height.

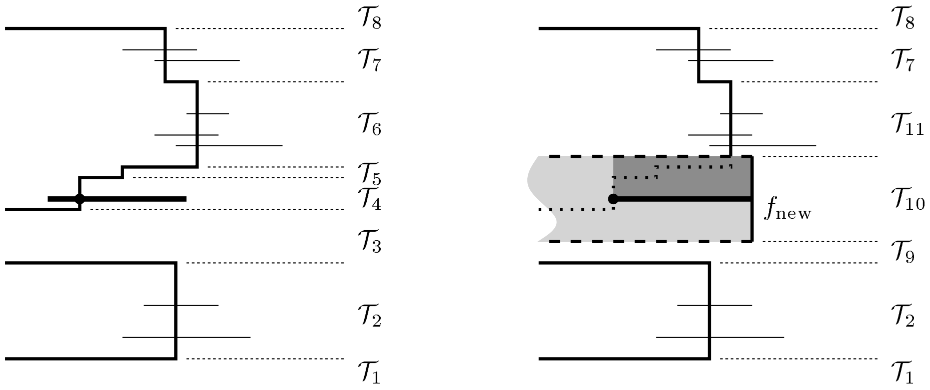


Figure 9: When the fat horizontal segment from H_{int} is chosen, the frontier changes by adding the fat vertical segment f_{new} shown right. The new label is dark grey. The grey range (light and dark) is the one with which queries in the priority search trees are done.

Update assistance structures.

1a. To determine which segments from H_{right} become part of H_{int} or H_{left} when the frontier changes we use a priority search tree on the left endpoints of segments in H_{right}. After the frontier changes with f_{new}, we query with the region left of f_{new} (grey in Figure 9) to locate all left endpoints of segments that are no longer in H_{right}. We delete these endpoints from the priority search tree, and we delete de corresponding segments from the heap $\mathcal{H}_{\text{int}}$. For each deleted segment we test if the right endpoint is right of the frontier. If so, that segment is in H_{int} and we insert it in the data structures for H_{int}. If not, the segment is in H_{left} and can be discarded.

2a. To determine which segments from H_{int} become part of H_{left} when the frontier changes we use a priority search tree on the right endpoints of segments in H_{int}. After the frontier changes with f_{new}, we query with the region left of f_{new} (grey in Figure 9) to locate all right endpoints of segments that have moved from H_{int} to H_{left}. The trees $\mathcal{T}_i$ and the heap $\mathcal{H}_{\text{int}}$ are updated by deleting the corresponding horizontal segments. We also delete the right endpoints we found from the priority search tree.

When the frontier changes, we must also reorganize the red-black trees and heap as a whole. Recall that we use a red-black tree for each vertical segment of F. At most three new vertical segments can arise when the frontier changes, but many more vertical segments may be destroyed. We use the red-black trees of the destroyed vertical segments of F to assemble the at most three new red-black trees. This is done by the operations SPLIT and CONCATENATE, which are standard for red-black trees. In Figure 9 the trees $\mathcal{T}_3$, $\mathcal{T}_4$, $\mathcal{T}_5$, and $\mathcal{T}_6$ are reorganized to the new trees $\mathcal{T}_9$, $\mathcal{T}_{10}$, and $\mathcal{T}_{11}$. The heap $\mathcal{H}_{\text{int}}$ is updated by removing the value of each destroyed tree, and inserting the value of each new tree.

3a. We don't need any additional data structures to update the heap on the vertical segments. However, we need to discover whether an extracted minimum from the heap really is in $V_{\text{int,right}}$. We use an augmented red-black tree for this test. It stores the vertical segments of the frontier sorted from bottom to top in the leaves. Each leaf also stores the x-coordinate of the vertical segment of F. Each internal node is augmented with a value that represents the minimum x-coordinate in its subtree. For each y-interval, the augmented red-black tree reports the minimum x-coordinate in this y-interval.

The whole algorithm is given below. Due to lack of space, details have been omitted.

Algorithm.

1. While there are still segments in the heap with V', or the structures for H_{right} or H_{int}, do the following steps.

2. Determine the minimum in the heap storing V'; assume the corresponding vertical segment is v. Search in the augmented red-black tree on F with v to see if it has some point right of F. If not, remove v from the heap and repeat this step.

3. Determine from the three heaps (for V', H_{right}, and H_{int}) which of them contains the smallest minimum in the root. Remove this minimum from the heap. Let l_i be the label position of point p_i corresponding to this minimum. Choose l_i as the next label to be placed.

4. Determine f_{new}, the right edge of l_i extended one unit downwards. Update the frontier F with f_{new}. Update the augmented red-black tree from **3a** with f_{new}. Search with the region horizontally left of f_{new} (grey in Figure 9) in the priority search trees of **1a** and **2a** and update the structures of **1**, **1a**, **2**, and **2a** accordingly.

Analysis. The basic structures used by the algorithm are heaps, red-black trees, augmented red-black trees, and priority search trees. All of these structures require $O(n)$ space for a set of size n. Also, these structures can be updated in $O(\log n)$ time per insertion or deletion, or extract-min for heaps. Red-black trees allow SPLIT and CONCATENATE in $O(\log n)$ time. The queries on the red-black trees take $O(\log n)$ time, and the queries on the priority search trees take $O(k + \log n)$ time, where k is the number of points found in the query range.

The efficiency of the algorithm follows from a couple of observations that can be shown easily. Any vertical segment f_{new} creates one vertical edge in the frontier F, and changes at most two of them. It follows that throughout the whole algorithm, at most $3n - 2$ different vertical edges appear in F. Therefore at most $3n-2$ vertical edges can be destroyed. This bounds the total number of red-black trees from **2** that can appear, the total number of SPLIT operations, and the total number of CONCATENATE operations by $O(n)$. Since SPLIT and CONCATENATE operations take $O(\log n)$ time each, at most $O(n \log n)$ time is spent on splitting and concatenating. The augmented red-black tree of **3a** can also be maintained in $O(n \log n)$ time for the same reasons.

For each new label placed, one query is done on each of the two priority search trees. Such a query takes $O(k + \log n)$ time, where k is the number of points in the range. These points are always deleted from the priority search tree, so the algorithm cannot spend time on reporting these points again later in the algorithm. The priority search trees are initialized with one point for each horizontal segment, and we never add more points to them. So in total, at most $O(n \log n)$ time is spent for querying in and updating the priority search trees.

We conclude:

Theorem 1 *Let P be a set of n points in the plane, and with each point a rectangular label with fixed height and some given width should be placed. An $O(n \log n)$ time and linear space algorithm exists to place at least half the maximum number of labels in any of the fixed-position and slider models.*

Remark 1. For the 1-slider and 2-slider models, we can omit the data structures listed under **3** and **3a**.

Remark 2. For fixed position models, the algorithm can be implemented using only one priority search tree and one heap. We initialize the priority search tree with the reference points of all labels that may still be placed, and the heap with their sums of the x-coordinate and the width. When a new label is chosen as the minimum in the heap, we query in the priority search tree with the appropriate range to find the reference points that are no longer possible. We remove these from the heap and from the priority search tree, and repeat by selecting the minimum from the heap.

Remark 3. We can extend the algorithm to labels of varying height. The approximation factor is $\frac{1}{1+\alpha}$, where α is the ratio of height of the highest and least high label. The running time in the fixed-position models remains $O(n \log n)$ and the space usage remains linear. The extensions are given in the full paper.

4 Polynomial time approximation scheme

A polynomial time approximation scheme for a labeling problem means that for any constant $\epsilon > 0$, there is an algorithm that runs in $O(n^c)$ time and places at least a fraction of $1 - \epsilon$ of the number of labels in an optimal placement. Here c is a constant that may depend on ϵ. Polynomial time approximation schemes (PTAS) have been developed for problems that are NP-hard, so all known algorithms producing an optimal solution require exponential time. A good survey of approximation algorithms for NP-hard geometric problems is by Bern and Eppstein [3].

For a set of unit squares, Hunt et al. [13] gave a PTAS to find the largest size subset of squares that don't intersect. This implies that there is a PTAS for labeling as many points as possible in any fixed-position model. This result was extended by Agarwal et al. [1] to labels with unit height but arbitrary width. In this section we show that the same result holds for the slider models. The algorithm seems only of theoretical interest, so we sketch the ideas only briefly. First we outline the approach for the 1-position model and unit height labels [1]. Then we extend to the 1-slider model and other models.

Let P be a set of n points and assume the 1-position model. We begin by stabbing the set of n possible labels by horizontal lines at distance greater than 1, and such that each label is stabbed by some horizontal line. Let these lines be $l_1, \ldots, l_m$ from top to bottom, and let $P_1, \ldots, P_m$ be the subsets such that P_i contains the points with labels intersecting l_i. The idea is to discard every t-th subset and solve the problem for $P_1, \ldots, P_{t-1}$ optimally, then solve the problem for $P_{t+1}, \ldots, P_{2t-1}$ optimally, and so on. These optimal solutions to subproblems can simply be joined to an optimal solution for $P_1, \ldots, P_{t-1}, P_{t+1}, \ldots, P_{2t-1}, \ldots$, since no label of a point in $P_1, \ldots, P_{t-1}$ can intersect a label of a point in $P_{t+1}, \ldots, P_{2t-1}$. For the whole set P, these joined optimal solutions to subsets form a suboptimal solution. The labeling problem for $t-1$ consecutive subsets can be solved optimally by dynamic programming in time polynomial in n but exponential in t.

Then we apply the Shifting Lemma idea [12] and discard the subsets P_{t-1}, P_{2t-1}, $P_{3t-1}, \ldots$, and compute an optimal solution for the remaining subset. Then we discard P_{t-2}, P_{2t-2}, $P_{3t-2}, \ldots$, and so on. One of the t solutions that we find is a $\frac{t-1}{t}$-approximation of the optimal solution for the whole set by the pigeon hole principle. For any constant t we have a polynomial time algorithm, so we obtain a polynomial time approximation scheme.

Assume now the 1-slider model. The idea of stabbing the labels by horizontal lines still works, and also the idea of combining the solutions for $P_1, \ldots, P_{t-1}$ and $P_{t+1}, \ldots, P_{2t-1}$ and so on. But we don't know how to solve the problem for $P_1, \ldots, P_{t-1}$ optimally in time polynomial in n. So the idea is to *approximate* the optimal solution for $P_1, \ldots, P_{t-1}$. Roughly, we set $k = t^2$ to be another constant, and determine the *leftmost* labeling of k labels from $P_1, \ldots, P_{t-1}$ by brute force. Since k is constant, we can test every subset of k labels and put them in a leftmost placement. If we cannot label the k points of some subset without intersection, we discard the subset. We choose the subset that gives the labeling that is leftmost among all subsets without intersections.

Let l be the leftmost vertical line that has the interiors of the k chosen labels to its left. We discard all label positions that intersect l or lie left of it. Then we repeat to find the next k leftmost labels. The observation is that the optimal solution cannot have a line left of our leftmost line with k labels to its left. So from the optimal solution, we discard at most one label from each of the $t-1$ subsets in our approximation. For the subsets $P_1, \ldots, P_{t-1}$, we approximate the optimal solution with a factor of $\frac{k}{k+t-1}$. The Shifting Lemma idea of discarding every t-th subset approximates the optimal solution with a factor of $\frac{t-1}{t}$. So the approximation factor for the whole algorithm is the product of the two: $\frac{k}{k+t-1} \cdot \frac{t-1}{t} = \frac{t^2}{t^2+t-1} \cdot \frac{t-1}{t} = \frac{t^2-t}{t^2+t-1}$. By increasing the value of t we get an approximation factor that is arbitrarily close to 1.

Assume that $P_1, \ldots, P_{t-1}$ together contain n' points. To find the leftmost k labels we try every $\binom{n'}{k}$ subsets of k points. We label these from left to right, and leftmost. This is easy in $O(k^2)$ time.

In the 4-slider model, we take the same approach, but it takes more time to find the leftmost labeling of k points. For each such subset, we test all possibilities of taking the top, bottom, left, or right slider for each label. This gives 4^k possibilities for one subset. Top and bottom sliders are put leftmost; left and right sliders are put bottommost. We must also try every one of the $k!$ insertion orders of the k labels. So testing one subset can be done in $O(4^k \cdot k! \cdot k^2)$ time.

Theorem 2 *For each of the slider models and for any constant $\epsilon > 0$, there is a polynomial time algorithm that labels at least $1-\epsilon$ times the maximum number of input points that can be labeled.*

5 Implementation and test results

The greedy algorithms of Section 3 for the fixed-position and slider models have been implemented and tested on several data sets. In this section we compare experimentally how many labels are placed in each of the six models.

The implementation of the algorithm was done in C++. For the data structures we made use of the LEDA library [18]. We have used a simplified implementation in three respects when compared to the one described here. The red-black trees $\mathcal{T}_i$ of **2** can be expected to contain only a few horizontal segments of H_{int} at any moment. So we used simple lists for them. LEDA doesn't have an implementation for priority search trees; we used orthogonal range trees instead. Thirdly, the augmented red-black tree doesn't profit much from the augmentation in practice. When searching for the minimum x-coordinate of F in a y-interval, we simply scan all leaves of the red-black tree in that interval. One can expect to visit only a few leaves, since the y-interval is only twice the unit height.

The first of the three data sets contains 1000 cities of the U.S.A. that must be labeled with their name. We used several different font sizes, and determined the bounding boxes of the text as the rectangles. The results are shown in Table 1. The codes 1P, 2P, and 4P are shorthand for the 1-, 2-, and 4-position models. The codes 1S, 2S, and 4S are shorthand for the slider models. The values in the second table show the percentages with respect to the 4-position labeling.

The second data set contains the 236 points of a data posting. The labels are measurement values and come from a book on geostatistics [15]. Figure 10 shows the labeled data set and the number of labels placed in each model.

The third data set contains the 75 points of a regression analysis. Here the points are clustered near a regression line, and the labels are simply identification numbers. Figure 11 shows the labeling.

The bottom tables of Figures 10 and 11 show that the 4-slider model sometimes places 10–15% more labels than the 4-position model. This improvement is significant, since it is always caused by a better labeling

font	1P	2P	4P	1S	2S	4S
					No. of labels placed	
5	851	950	971	990	993	999
6	777	910	952	967	982	986
7	705	852	901	932	964	972
8	686	845	896	918	952	958
9	607	758	817	836	890	902
10	554	704	769	787	853	872
11	520	657	721	735	805	831
12	500	637	709	719	796	813
13	448	570	638	649	716	734
14	433	557	624	637	695	712
15	382	494	550	556	627	645

font	1P	2P	4P	1S	2S	4S
					Percentage w.r.t. 4-position model	
5	87	97	100	101	102	102
6	81	95	100	101	103	103
7	78	94	100	103	106	107
8	76	94	100	102	106	106
9	74	92	100	102	108	110
10	72	91	100	102	110	113
11	72	91	100	101	111	115
12	70	89	100	101	112	114
13	70	89	100	101	112	115
14	69	89	100	102	111	114
15	69	89	100	101	114	117

Table 1: 1000 cities on a large map.

of the areas that are difficult to label. We also created artificial, pseudo-random data sets where all areas are difficult to label. Here we indeed found higher improvements: up to 92%.

Efficiency was not the main motivation for these experiments. Still it appeared that the label placement was computed in a few seconds for all data sets we tried, up to 2500 points. A plot shown on a computer screen seldomly contains more than 1000 labeled points.

6 Conclusions and extensions

We have compared six different models for labeling a set of points with rectangular labels of fixed height. New in our paper (with the exception of Hirsch's work [11]) is that we allow labels to be placed anywhere while touching the point. Previous papers limited the number of possible positions for a label to a constant. Firstly, we showed that these slider models for labeling can in some cases be considerably better than the fixed-position models. Secondly, we showed that for each of the slider models, there is a simple factor-$\frac{1}{2}$ approximation algorithm that requires $O(n \log n)$ time and linear space. Thirdly, we sketched a polynomial time approximation scheme for each of the slider models. Finally, we experimentally compared the six different models on various data sets, using the factor-$\frac{1}{2}$ approximation algorithms. We observed, for instance, that the 4-slider model can place 10–15% more labels than the 4-position model on real-world data. Improvements are higher for pseudo-random data.

The simple approximation algorithms we gave only apply to labels of the same height. We also have variations of our algorithms that work for labels of varying heights; these are described in the full paper.

We haven't compared our results with the extensive tests of Christensen et al. [4], since our main objective was to mutually compare the different models. Still it would be interesting to see how the 4-slider model performs compared to the various algorithms tested by Christensen et al. [4]. Since we don't have an exact algorithm to compute an optimal labeling, we don't know whether the factor-$\frac{1}{2}$ approximation algorithms really perform better than the guaranteed factor $\frac{1}{2}$.

Another extension of our work is to produce label placements that have high quality according to other criteria as well, like avoiding ambiguity, and giving preference to certain label positions over others.

References

[1] P.K. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. In *Proceedings 9th Canadian Conference on Computational Geometry*, pages 233–238, 1997.

[2] T.C. Bailey and A.C. Gatrell. *Interactive Spatial Data Analysis.* Longman, Harlow, 1995.

[3] M. Bern and D. Eppstein. Approximation algorithms for geometric problems. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 296–345. PWS Publishing Company, Boston, MA, 1997. ISBN 0-534-94968-1.

[4] J. Christensen, J. Marks, and S. Shieber. An empirical study of algorithms for point-feature label placement. *ACM Trans. Graph.*, 14:202–232, 1995.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[6] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry – Algorithms and Applications.* Springer-Verlag, Berlin, 1997.

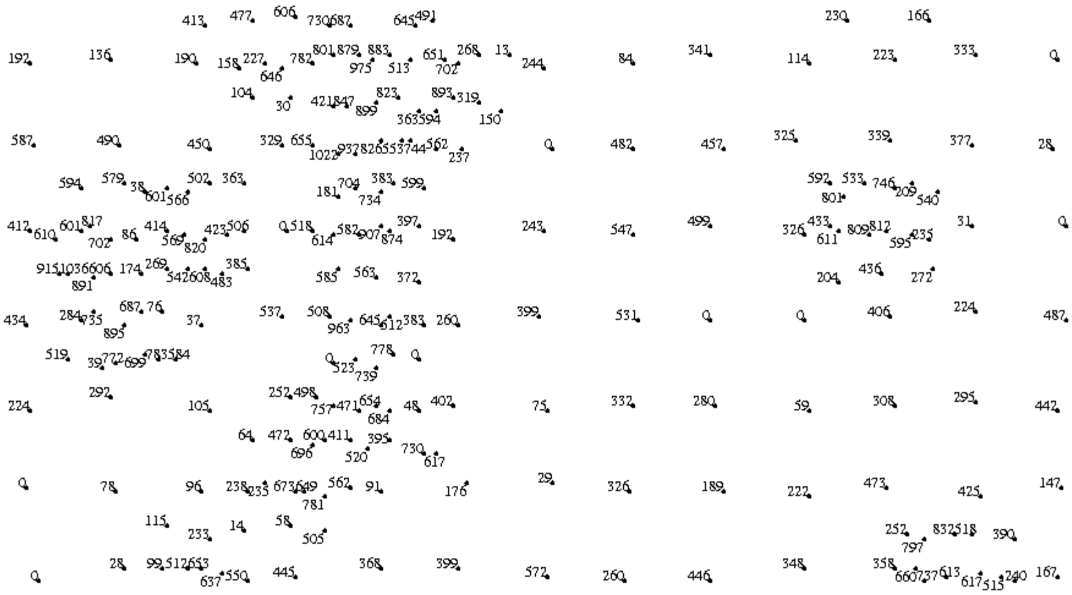
[7] S. Doddi, M. Marathe, A. Mirzaian, B. Moret, and B. Zhu. Map labeling and its generalization. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pages 148–157, 1997.

[8] J. S. Doerschler and H. Freeman. A rule-based system for dense-map name placement. *Commun. ACM*, 35:68–79, 1992.

[9] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 281–288, 1991.

[10] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inform. Process. Lett.*, 12(3):133–137, 1981.

[11] S.A. Hirsch. An algorithm for automatic name placement around point data. *The American Cartographer*, 8:5–17, 1982.



No. of labels placed

font	1P	2P	4P	1S	2S	4S
5	229	236	236	236	236	236
6	216	235	235	236	236	236
7	197	219	230	236	236	236
8	197	219	230	236	236	236
9	185	205	218	235	236	236
10	175	193	207	223	231	230
11	174	189	200	213	221	224
12	174	189	200	213	221	224
13	169	180	188	203	212	212
14	169	180	188	203	212	212
15	157	170	176	192	200	203

Percentage w.r.t. 4-position model

font	1P	2P	4P	1S	2S	4S
5	97	100	100	100	100	100
6	91	100	100	100	100	100
7	85	95	100	102	102	102
8	85	95	100	102	102	102
9	84	94	100	107	108	108
10	84	93	100	107	111	111
11	87	94	100	106	110	112
12	87	94	100	106	110	112
13	89	95	100	107	112	112
14	89	95	100	107	112	112
15	89	96	100	109	113	115

Figure 10: Labeling of the data posting in 9pt font using the 4-slider model (scaled to fit), and tables with the performance.

[12] D. S. Hochbaum and W. Maas. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, 32:130–136, 1985.

[13] H.B. Hunt III, M.V. Marathe, V. Radhakrishnan, S.S. Ravi, D.J. Rosenkrantz, and R.E. Stearns. A unified approach to approximation schemes for NP- and PSPACE-hard problems for geometric graphs. In *Proc. 2nd Europ. Symp. on Algorithms*, volume 855 of *Lect. Notes in Comp. Science*, pages 424–435, 1995.

[14] E. Imhof. Positioning names on maps. *The American Cartographer*, 2:128–144, 1975.

[15] E. H. Isaaks and R. M Srivastava. *An Introduction to Applied Geostatistics.* Oxford University Press, New York, 1989.

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

[17] E. M. McCreight. Priority search trees. *SIAM J. Comput.*, 14:257–276, 1985.

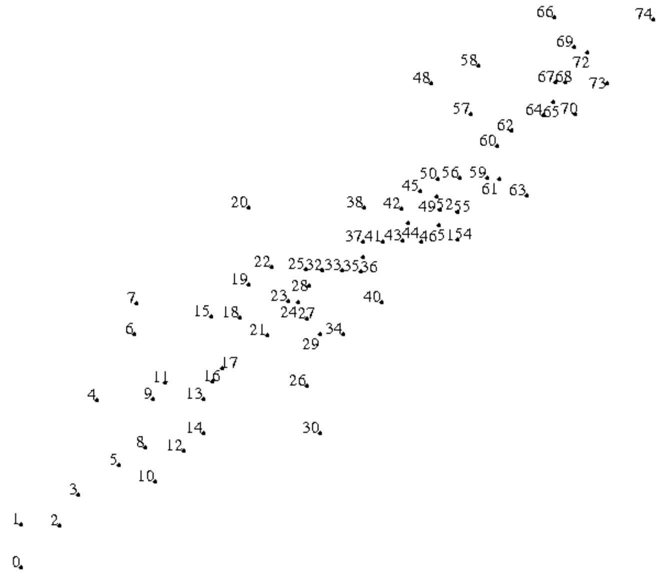
[18] K. Mehlhorn and S. Näher. LEDA: a platform for combinatorial and geometric computing. *Commun. ACM*, 38:96–102, 1995.

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

[20] Frank Wagner and Alexander Wolff. Map labeling heuristics: Provably good and practically useful. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 109–118, 1995.

[21] A. Wolff. A map labeling bibliography, 1997. `http://www.inf.fu-berlin.de/map-labeling/papers.html`.

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



No. of labels placed

font	1P	2P	4P	1S	2S	4S
5	75	75	75	75	75	75
6	75	75	75	75	75	75
7	70	74	74	75	75	75
8	70	74	74	75	75	75
9	60	69	70	73	74	74
10	58	65	68	72	72	72
11	55	61	66	66	70	70
12	55	61	66	66	70	70
13	51	58	64	63	68	71
14	51	58	64	63	68	71
15	50	56	61	62	67	68

Percentage w.r.t. 4-position model

font	1P	2P	4P	1S	2S	4S
5	100	100	100	100	100	100
6	100	100	100	100	100	100
7	94	100	100	101	101	101
8	94	100	100	101	101	101
9	85	98	100	104	105	105
10	85	95	100	105	105	105
11	83	92	100	100	106	106
12	83	92	100	100	106	106
13	79	90	100	98	106	110
14	79	90	100	98	106	110
15	81	91	100	101	109	111

Figure 11: Labeling of the scatterplot in 11pt font using the 4-slider model (scaled to fit), and tables with the performance.