Practical Extensions of Point Labeling in the Slider Model*

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

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

Abstract

This paper extends on research by the authors together with Alexander Wolff on point label placement using a model where labels can be placed at any position that touches the point (the slider model). Such models have been shown to perform better than methods that allow only a fixed number of positions per label. The novelties in this paper include respecting other map features that must be avoided by the labels, and incorporating labels with different height. The result is an efficient and simple $O((n+m)\log(n+m))$ time algorithm with a performance guarantee for label placement in the slider model. Here n is the number of points to be labeled and m is the combinatorial complexity of the map features that must be avoided. Due to its efficiency, the algorithm can be used in interactive and on-line mapping.

1 Introduction

The automated map labeling problem is a well-known problem for many years in cartographic and GIS research. Manual label placement is a time-consuming task and it is natural to try and automate it. This requires a good set of rules according to which a program can work. Such rules have been given decades ago by Imhof [9], Alinhac [1], and Yoeli [13]. From the algorithmic community, the label placement problem has received considerable attention more recently [2], [5], [11]. Usually, a strongly abstracted version of cartographic label placement is considered, the only rule surviving being the non-overlap of different labels. However, simple adaptations to these approaches can yield solutions that also avoid ambiguity, and have other desirable properties of a good labeling.

In the labeling of topographic maps there are three main types of label: point labels, linear feature labels, and area feature labels. Labeling a set of points has been studied more extensively than lines and areas. Point set labeling is regarded as an optimization problem, where the criterion is either maximum label (font) size, or maximum number of labels placed. Both problems are intractable (in most formulations)[6], and an efficient approximation algorithm is usually sought. An approximation algorithm computes a labeling that is guaranteed not

*This research is supported by the Dutch Organization for Scientific Research (N.W.O.) and by the ESPRIT IV LTR Project No. 21957 (CGAL).

to be much worse than the best labeling possible for the given point set. In practice the performance guarantee is usually rather pessimistic. The computation time needed to run the algorithm is an important issue, especially for interactive and on-line mapping. Label placement is an on-line problem, because usually label positions cannot be precomputed and stored in the database [7].

The *point set labeling problem* is: Given a set P of n points in the plane, and with each point a rectangle (the bounding box of the text), place as many of the n rectangles as possible such that each rectangle touches its point and no two of the rectangles overlap.

The map labeling bibliography on the internet [12] lists over fifty papers on the topic; we'll only mention a few results and techniques that are most relevant to our research. Nearly all papers that deal with automated label placement start out by considering only a finite number of allowed positions for each label. For example, in the 4-position model, a label may be placed such that one of the four corners of the bounding box of the label coincides with the point to be labeled. In the 8-position model, the label may also be such that the point to be labeled lies in the middle of any of the four edges of the bounding box. These finite position models give an easy discretization of the solution space, which facilitates the development of algorithms that actually try to place the labels without overlap.

Hirsch [8] was probably the first to allow any label position that touches the point to be labeled. His approach is based on defining repelling forces between labels, and iterative methods to converge to label positions that have little overlap. His method, however, doesn't give any guarantee on the number of labels that will be placed, nor on the running time. The idea of allowing any touching label position was pursued further by van Kreveld, Strijk, and Wolff [10]. They called it the sliding label model, as opposed to the fixed position model. In that paper it was analyzed how much benefit could be obtained from the extra flexibility of sliding label positions, both in theory and in practice. At the same time, that paper showed that a relatively simple implementation of labeling in the sliding label model is possible. A factor-2 approximation algorithm was given that runs in $O(n \log n)$ time. On the tested data sets it appeared that the sliding label models allowed more labels to be placed than the fixed position models. This holds true even if sophisticated, time-consuming optimization strategies like simulated annealing are used for fixed position labeling. The simple implementation of the sliding label model outperformed the simulated annealing approach of Christensen et al. [2] by up to 10% in number of labels placed, and in orders of magnitude in running time.

The research in [10] mainly focused on the comparison of the different labeling models, and whether sliding models are to be preferred over fixed position models. Therefore, two important practical aspects of sliding label models were not considered. Firstly, all labels were considered to consist of text with the same height. In other words, different font sizes were not allowed. Secondly, it was assumed that the point set to be labeled was free of context. No features needed to be shown other than the points and their labels. For the labeling of geostatistical data in scatterplots, these restrictions may be valid, but for most other point labeling problems one has to deal with different font sizes and other objects to be visualized than only labels and points. This paper addresses both of these extensions to [10].

When there are other map features to be avoided, it is very useful to have a sliding label model, because it can easily happen that none of the four positions in a 4-position model can be used without overlap. This can be seen from

Figure 1: Enlarged example of labeling in the sliding label model with obstacles.

the cities South Bend and Erie in Figure 1. In the figure, one can see that label positions haven't been optimized for preference positions, aesthetic criteria haven't been used, and ascenders and descenders of the letters haven't been considered. This would be necessary in a real application.

In Section 2 we assume that a point set to be labeled is given, together with a set of line segments that may not be intersected by the labels. For statistical applications, the line segments can be the two axes of a regression analysis plot together with the regression line. For cartographic applications, the set of line segments can be administrative boundaries like the boundaries of the States of the U.S.A. We present an algorithm that ensures that none of the labels intersect the line segments in the given set (and of course, they won't overlap each other). At the same time, the method ensures that a point and its label will be on the same side in the case of a regression line or State boundaries. The algorithm is in fact a simple preprocessing step that runs in $O((n+m)\log(n+m))$ time for n points to be labeled, and m line segments to be avoided. After the preprocessing, one could select a few label positions and run a fixed-position labeling method, or one could use the algorithm of van Kreveld, Strijk and Wolff for sliding labels. In either case, a factor-2 approximation algorithm is obtained that runs in $O((n+m)\log(n+m))$ time overall. The method automatically extends to avoiding whole regions (including interior) instead of only line segments. At the end of the section we give some sample output of our algorithm.

In Section 3 we consider how the sliding label algorithm from [10] can be adapted to incorporate labels of different height. If r different label heights are used, the algorithm we present takes $O(rn\log n)$ time. However, this algorithm is of restricted interest. A label number maximization approach tends to choose small (short and not high) labels over bigger ones. So labels in big fonts may be the first to be omitted, although common sense tells that these are more important.

Section 4 discusses ways to resolve the problem just mentioned. Three approaches may come to mind. Firstly, we could try and place the larger labels at

the expense of the smaller labels after the standard algorithm has completed. Such a patching up approach is not very elegant. Secondly, we can place labels in the order of decreasing height. We run one of the existing algorithms for all labels of the largest height, and add these as obstacles for the next run of the algorithm. In Section 2 we already discussed how to handle obstacles. Thirdly, we can adapt the sliding label algorithm and make different choices when the next label is chosen. The last two approaches are compared to each other on two test sets. The conclusions of this paper can be found in Section 5.

2 Point labeling amidst obstacles

In this section we are given m segments $s_1, \ldots, s_m$ and n points $p_1, \ldots, p_n$, each with a bounding box of its text. This bounding box is called the label. The segments may only touch at their endpoints. We want to determine for each point feature all label positions belonging to that point feature that don't intersect the segments $s_1, \ldots, s_m$. Once this is known, one of the existing label placement algorithms can be used. As in van Kreveld et al.[10], the label positions are represented as the positions of a reference point on the label. We choose the lower left corner of the label as the reference point. All positions of the label that touch its point are represented by four segments on which the reference point should lie. The idea of point labeling amidst obstacles is to precompute all positions of the reference point that cannot be used due to the line segments $s_1, \ldots, s_m$. This implies truncating the segments on which the reference point must lie. We first give a brute force $O(nm)$ time solution, then we improve it to $O((n+m)\log(n+m))$ time.

2.1 The algorithm

Denote the coordinates of a point p_i by (x_i, y_i), the width of its label by w_i and the height of its label by h_i. We only look at the case that the bottom edge of the label contains the point. The other cases in which the point is on the left, top, or right edge of the label are handled in a similar way, either by translating the point downwards or by interchanging x and y coordinates.

The position for the label with its bottom edge containing the point ranges between two extremes. The left extreme occurs when the bottom right corner of the label coincides with the point, the right extreme occurs when the bottom left corner coincides with the point.

Observe that the label always contains the vertical segment $v_i = \{x_i\} \times [y_i, y_i + h_i]$. The first check we make is to test whether there is a segment s_j that intersects v_i. If so, we know that the label cannot be placed above the point feature.

The next step is to determine how far the label can be shifted to the right starting from the left extreme until it hits a segment to the right of v_i (here we don't care whether the label intersects a segment to the left of v_i). Similarly we test how far the label can be shifted starting from the right extreme to left until it hits a segment to the left of v_i. We now deal with shifting to the right (shifting to the left is handled analogously). We distinguish two cases, see Figure 2:

1. The label hits the interior of a segment.

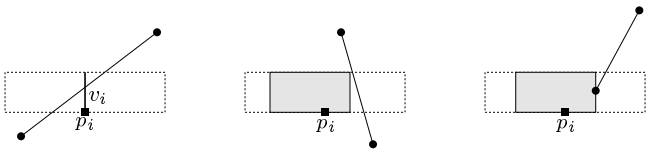

Figure 2: In the left picture the point cannot be labeled above it. In the middle picture case 1 is shown and the right picture case 2.

2. The label hits an endpoint of a segment.

In the first case the label hits the segment at the top right corner (case 1a) or the bottom right corner of the label (case 1b). For each point p_i we test for every segment s_j whether we are in case 1a, case 1b, or case 2, and compute the x-coordinate for each case where it hits the segment. The leftmost x-coordinate of these is called x_{ij}. Then we determine the leftmost x_{ij} for all segments $s_1, \ldots, s_j$. This is the x-coordinate where the label first hits a segment to the right of v_i.

After we have computed how far the label can be shifted to the left and to the right until it hits a segment we know the interval in which the label must lie. If the length of this interval is less than w_i then there is no label position above the point feature that does not intersect a segment. Otherwise, we determine the segment representing the valid positions for the reference point of the label.

The running time of this algorithm is $O(nm)$ since for every point feature we test all segments for restricting label positions. For one segment and one point these tests can be done in constant time.

When the number of segments m is large we can improve the $O(nm)$ time algorithm. This is done as follows. We build four data structures. The first one is used to detect the situation where line segment v_i intersects one of the line segments s_j. This data structure is a vertical decomposition of all segments $s_1, \ldots, s_m$, which is preprocessed for efficient planar point location [4]. With this data structure we can perform the check whether a segment intersects the segment v_i in $O(\log m)$ time. We perform a point location query with the point p_i in the vertical decomposition. The top edge of the cell containing p_i is the first segment above p_i that intersects the vertical line containing v_i. If the segment intersects v_i then we cannot place a label above p_i. Building the point location structure for the vertical decomposition takes $O(m \log m)$ time and $O(m)$ space. The point location query takes $O(\log m)$ time.

The second data structure is used to handle cases 1a and 1b. It is a horizontal decomposition of all segments $s_1 \ldots, s_m$, also preprocessed for efficient planar point location. We do a point location query with point p_i and point $p_i + (0, h_i)$ in the horizontal decomposition. The right edge in the cell containing p_i is the first segment that is hit by the bottom right corner of the label if it is shifted to the right starting from the left extreme. Similarly, the right edge of the cell containing $p_i + (0, h_i)$ is the first segment that is hit by the top right corner of the label. Building the horizontal decomposition takes $O(m \log m)$ time and $O(m)$ space. Point location requires $O(\log m)$ for each query.

The third and fourth data structures handle case 2. The third data structure is used for finding for each point p_i the first segment endpoint to the left of v_i.

Symmetrically the fourth data structure is used for finding for every point p_i the first segment endpoint to the right of v_i. Because of the symmetry of the two subcases we only describe the test for segment endpoints to the left of v_i. The data structure we use is a priority search tree [4]. We would like to answer for each point p_i the following query: What is the rightmost segment endpoint in the halfstrip $(x < x_i) \wedge (y_i \leq y \leq y_i + h_i)$, the area to the left of v_i ? Since a priority search tree can only find the rightmost segment endpoint in a strip $y_i \leq y \leq y_i + h_i$, we must use a trick. We ensure that at the query moment the priority search tree only contains segment endpoints with x-coordinate less than x_i. In that way the rightmost segment endpoint of the strip $y_i \leq y \leq y_i + h_i$ is also the rightmost segment endpoint in the halfstrip $(x < x_i) \wedge (y_i \leq y \leq y_i + h_i)$. To ensure that the priority search tree only contains points with x-coordinate less than x_i at the moment we want to answer the query for p_i, we do the following. The points are tested in increasing order of x-coordinate, and for a point p_i we first insert into the priority search tree the segment endpoints which have x-coordinate less than x_i and were not already in the priority search tree. This can easily be determined by sorting in advance the segment endpoints for increasing x-coordinate and keeping a pointer to the first segment endpoint that was not yet inserted into the priority search tree. Sorting the points requires $O(n \log n)$ time, and sorting the segment endpoints takes $O(m \log m)$ time. Inserting a segment endpoint into the priority search tree takes $O(\log m)$ time per point. Since we insert each segment endpoint once this adds up to $O(m \log m)$ time. Computing the rightmost point in a strip takes $O(\log m)$ time.

The four data structures together answer the question whether a point p_i can be labeled with the label above it, and also the maximum extent leftwards and rightwards of this label. If the data structures are present, this takes only $O(\log m)$ time for each point in $p_1, \ldots, p_n$. So the total query time is $O(n \log m)$ and the total time needed to construct the data structures is $O(m \log m + n \log n)$. This results in a solution based on efficient data structures that takes $O((n + m) \log(n + m))$ time and linear space.

When the maximum extents with respect to the obstacle line segments are known for each label, there are two ways to proceed. Either we select a fixed number of candidate positions from the ones still possible and apply a fixed position method for label placement, or we use the truncated segments for the reference point directly in the sliding label algorithm.

2.2 Experiments

We implemented the algorithm and tested it on two data sets. The first is a scatterplot of 75 points in a regression analysis. The obstacle to be avoided is the regression line. The labels are the sequence numbers of the data points; labels 24, 36, 49, and 56 could not be placed. See Figure 3. The second data set consists of the U.S.A. with state boundaries. The method guarantees that the label of any city appears in the same state as the city. The method also makes sure that if there is a label position touching the point which doesn't intersect boundaries, then that label position is considered in the algorithm. The point may not be labeled due to other labels, though. Figure 4 shows part of the U.S.A. with labels. We have not attempted to place the labels at the best position possible from the aesthetic point of view. If the algorithm is used

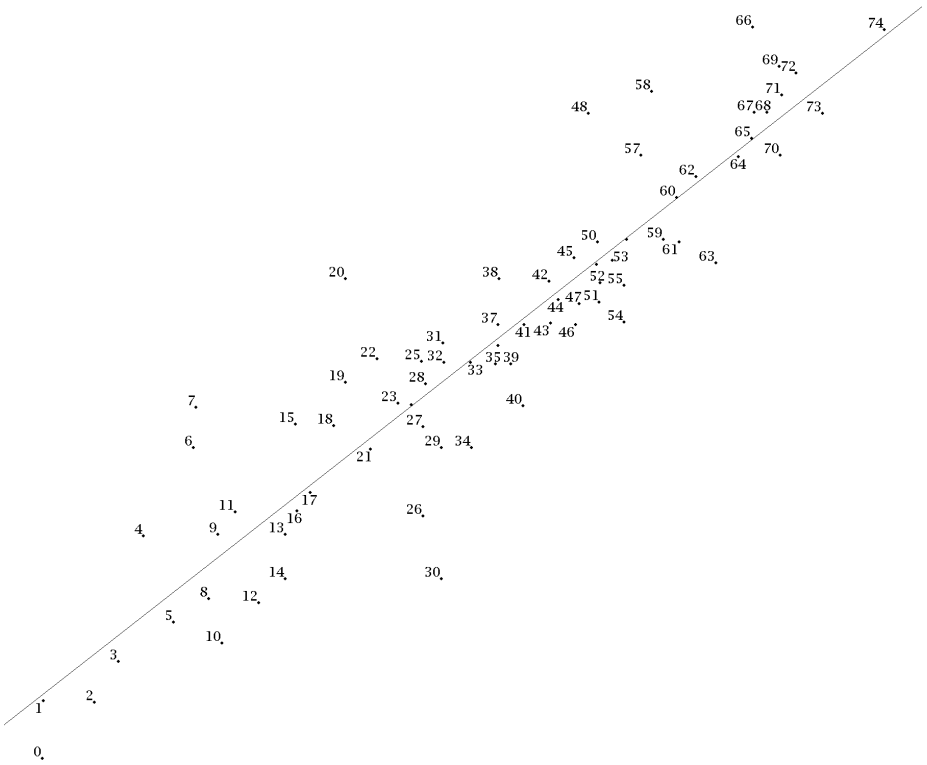

Figure 3: Scatterplot of data points in a regression analysis. The labels are placed such that they don't overlap the regression line.

in an application that requires high quality, then a post-processing step must be included to slide labels from the computed positions into better positions, taking the descenders and ascenders of the letters into account.

3 Point labeling for point labels of varying height

In this section we describe an algorithm for point feature labeling with sliding labels. If the number of different label heights of the point features is r then the running time of the algorithm is $O(rn \log n)$. Since on maps the number of different label heights usually is less than 10, we can consider r a constant that doesn't depend on n.

The algorithm we use is an extension of the algorithm described in van Kreveld et al. [10] that computes a labeling with sliding labels when all labels have the same height. We start by describing the main principles of that algorithm.

The labels must be of fixed height but arbitrary width. We describe an $O(n \log n)$ time algorithm for the slider model that approximates an optimal solution in the following sense. If the maximum number of labels that can be placed is k_{opt}, then our algorithm places at least $k_{\text{opt}}/2$ labels: a factor-2 approximation algorithm. In most data sets, however, we expect to come much closer to the optimum.

Figure 4: Map of the south eastern part of the U.S.A. The labels are placed such that they don't overlap the state boundaries.

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 label candidates of unlabeled points and that does not intersect previously placed labels.

Lemma 1 *Given a set of points and with each point, a label of the same height, 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 for the slider model.*

Proof: See proof in [10]. ■

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 bottom left corner. The possible positions of the reference point of a point p_i are represented by four line segments. Two are horizontal, $s^i_{h,\text{top}}$ and $s^i_{h,\text{bottom}}$, and two are vertical, $s^i_{v,\text{left}}$ and $s^i_{v,\text{right}}$. 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 1. We can always scale the y-coordinates 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. The open rectangle that exactly covers l_i and l'_i and their mutual bounding edge is the *extended rectangle* $\tilde{l}_i$. Since labels are placed from left to right, no reference point positions in nor to the left of $\tilde{l}_i$ will be accepted later by the algorithm. Suppose a subset of the points has already received labels by the algorithm.

The *right envelope* of all extended rectangles $\tilde{l}$ for all labels l outlines all reference point positions that are impossible, or cannot occur any more, see the bold line in Figure 5. We call this right envelope the *frontier* and denote it by F.

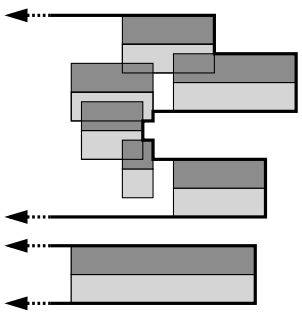

Figure 5: Frontier of the placed labels (dark grey) and their lowered copies (light grey).

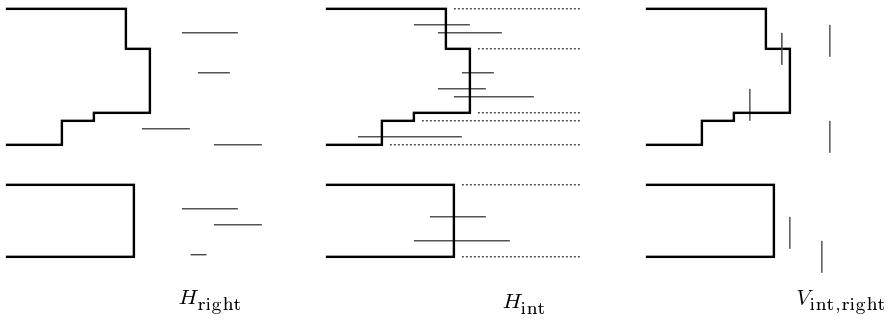

Figure 6: The sets H_{right}, H_{int}, and $V_{\text{int,right}}$. The dashed lines in the middle picture separate the segments of H_{int} that are restricted by different segments of F.

To determine the next leftmost label, we only have to consider the frontier F and the segments $s^i_{h,\text{top}}, s^i_{h,\text{bottom}}, s^i_{v,\text{left}}$, and $s^i_{v,\text{right}}$ of the points p_i to the right of F that don't have a label yet. Given a horizontal segment s_h and the frontier F, there are three possibilities: (i) s_h lies completely left of F. Then s_h can be discarded; a point on it cannot be a reference point for a label that doesn't overlap another label. (ii) s_h lies completely right of F. Then the leftmost point on s_h is a candidate for the next leftmost label. (iii) s_h intersects F. Then a point just right of the intersection point is the candidate. For a vertical segment s_v, a similar situation occurs. If s_v lies left of F, it can be discarded; if s_v lies right of F, any point on s_v can be chosen; and if s_v and F intersect, then any point on s_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 6. 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

efficiently, we need some 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 $\mathcal{P}$. These are described in standard textbooks on algorithms [3] and computational geometry [4]. Using these data structures the algorithm of [10] labels the points from left to right in $O(n \log n)$ time in total.

3.1 The algorithm

In the version of the labeling problem where r different label heights are allowed, we have to change the given solution. It is not sufficient any more to use only one frontier and the corresponding data structures. We describe the adaptations required next.

Assume we have r different label heights $h_1, \ldots, h_r$. We partition all points in classes P_j in which all points have labels with height h_j. Instead of one frontier as in the uniform height algorithm we now maintain r frontiers F_j, which we maintain to treat the points in P_j. We also have r sets H^j_{right}, H^j_{int}, and $V^j_{\text{int,right}}$ that store, respectively, the horizontal segments that intersect the frontier F_j, the horizontal segments that lie completely to the right of F_j and the vertical segments that do not lie completely to the left of F_j.

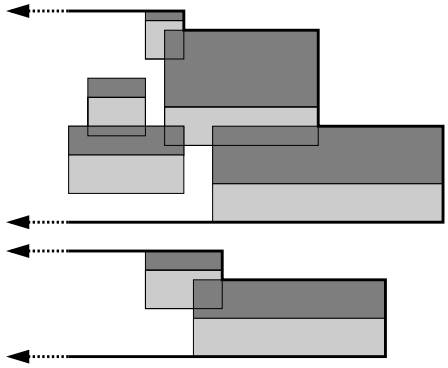

Figure 7: Frontier F_j of the placed labels (dark grey) and their corresponding rectangles with height h_j (light grey).

The r data structures are capable of giving the leftmost label l_j among the points in P_j as described in [10]. So we can determine the leftmost possible label of all points by taking the leftmost label l from $l_1, \ldots, l_r$. We place this label l and the next step is to update all frontiers $F_1, \ldots, F_r$ and the sets H^j_{right}, H^j_{int}, and $V^j_{\text{int,right}}$ for $j = 1, \ldots, r$.

Recall that the frontier of F_j is the boundary of the area where the reference points of the horizontal and vertical segments of the points P_j cannot lie any more. If a label l has been placed, then no reference point position inside l is possible. The same holds for reference points inside the rectangle l' that has the same width as l but height h_j and that is placed immediately below l. If the reference point of a label belonging to a point in P_j would lie in l' then the label would intersect l. So the frontier F_j is the right envelope of the placed

labels and the rectangles that have the same width as the labels but height h_j and that are placed immediately below their labels, see Figure 7.

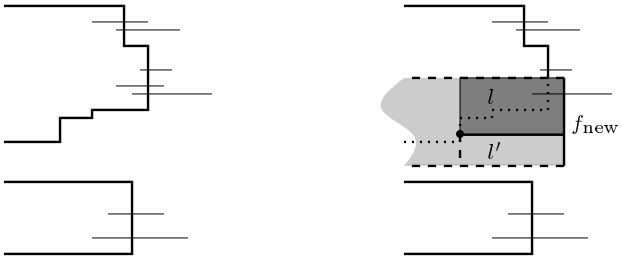

Figure 8: When the new label is chosen, the frontier F_j becomes the right envelope of f_{new} and the old frontier. The new label l is dark grey. The grey range (light and dark) is the area in which all reference points are removed.

So we must change every F_j by extending it with the label l and the rectangle l' below it, see Figure 8. Let f_{new} be the right edge of l and l'. Because the way the fixed height algorithm works does not depend on the height of f_{new} being a multiple of the height of l or h_j, we can compute the updated frontier and the sets H^j_{right}, H^j_{int}, and $V^j_{\text{int,right}}$ using the existing data structures for each j.

The running time is only a factor r worse since now we must update the data structures for the r classes of points instead of just one class. The algorithm needs linear space because the points are divided in sets $P_1, \ldots, P_r$ and the space needed is linear in the size of each P_j.

We conclude:

Theorem 1 *Let S be a set of n points in the plane, and for each point a rectangular label with a height and width is given. If only a constant number of label heights are used, then there is an $O(n \log n)$ time and linear space algorithm which places the labels according to the greedy sliding method.*

3.2 Approximation factor

For the fixed height algorithm we could prove that it was a factor-2 approximation algorithm. That means that if the maximum number of labels that can be placed is k_{opt}, then the fixed height algorithm places at least $k_{\text{opt}}/2$ labels.

If we drop the restriction that all labels have the same height, we do not have the factor-2 any more. But we can prove another approximation factor. Let $h_{\min}$ be the minimum height among all the labels, and $h_{\max}$ the maximum height among all labels and $\alpha = \frac{h_{\max}}{h_{\min}}$. Then we can prove that the approximation factor of the algorithm is $1 + \alpha$. Note that this is line with the fixed height approximation factor because in that case $\alpha = 1$.

The proof relies on the fact that if we place a leftmost label of height $h_{\max}$ then, instead of this label we could have placed at most $\alpha + 1$ labels of height $h_{\min}$. If we extend this to all labels we get the factor-$(\alpha + 1)$ approximation.

Note that these approximation factors are only guarantees that in the worst case we can label at least a certain fraction of the maximum possible. In practice the algorithm performs much better. For example, the fixed height algorithm places often more labels than the standard simulated annealing algorithm.

4 Priority of labels

The algorithm described above tends to choose short labels over longer ones. Since in general labels with a larger font size also have a longer label, they may not be chosen although in practice such labels are more important than labels in small fonts. To resolve this problem we suggest three possible approaches. Firstly we could use a post-processing approach in which first the algorithm above is run, and after that, important labels are added by shifting or removing existing labels. Such an approach is not very elegant because it tries to fix things afterwards instead of taking them into account in an earlier phase.

In the second method we first run the algorithm only on the labels with the largest height. The labels of the first run become obstacles for the subsequent runs. Then the algorithm is run on the labels with the second largest height and we add the placed labels as obstacles for the next runs, and so on. In Section 2 we already discussed how to handle obstacles. The overall running time of the algorithm is $O(n \log n)$ if a constant number of different heights are used. A disadvantage of this method is that labels placed in earlier runs don't take into account the possible position of labels in the later runs at all. Placement of the big labels could result in the placement of only few smaller labels, whereas many of them would fit if the big labels were placed differently.

The last method uses an adapted version of the algorithm given in the previous section. For each point we have the real label with a certain height and width. We will also use an imaginary label that has the same height as the normal label, but its width may be smaller than that of the real label. On an abstract level, the algorithm in Section 3.1 has two parts. The first is selecting the leftmost label, and the second is updating the set of reference points that belong to labels that are still allowed. For the first part we use the imaginary labels and for the second part the real labels. Thus we use the imaginary labels for computing the leftmost label. We will make labels with a large height have a much smaller imaginary label, so they will be chosen sooner than other labels. For any label chosen, the real size of it is used to update the frontiers F_j and the sets H^j_{right}, H^j_{int}, and $V^j_{\text{int,right}}$. Placing a label of a certain point can be given a higher priority by giving it an imaginary label with a small width. One can show that this adaptation of the algorithm does not affect the running time of $O(n \log n)$.

4.1 Experiments

We performed experiments to compare the labelings produced by the second method and by the last method. The test set consisted of 156 cities of the U.S.A. Three different fonts were available for the labels. The 6 cities with a population of one million or more were assigned a label in the largest font. The 15 cities with a population of more than half a million got middle sized labels. The remainder of the cities (135) were to be labeled with the smallest font. The imaginary width of the labels was determined by the font size. This width was 0 for the largest font labels, one tenth of the real label width for the middle sized font, and the imaginary width equaled the real label width for the smallest labels. In the first test the map was scaled such that we could place roughly 50 labels. In the second test we scaled the map to allow roughly 100 labels. The results are shown in Tables 1 and 2.

	labels placed			
method	total	smallest font	middle font	largest font
font-after-font	50	36	9	5
imaginary width	49	33	10	6

Table 1: Comparison of the two methods on a map with roughly 50 labels.

	labels placed			
method	total	smallest font	middle font	largest font
font-after-font	97	79	12	6
imaginary width	100	79	15	6

Table 2: Comparison of the two methods on a map with roughly 100 labels.

It follows from the tables that the imaginary width method places at least as many largest font labels as the font-after-font method and moreover, more middle font labels.

From the Figures 9 and 10 showing the labeling for the 100 label test, one can see that in the imaginary width method the label of Columbus was placed, but in the font-after-font method it was not, because the label of Philadelphia was placed as much to the left as possible, unaware of the future label of Columbus.

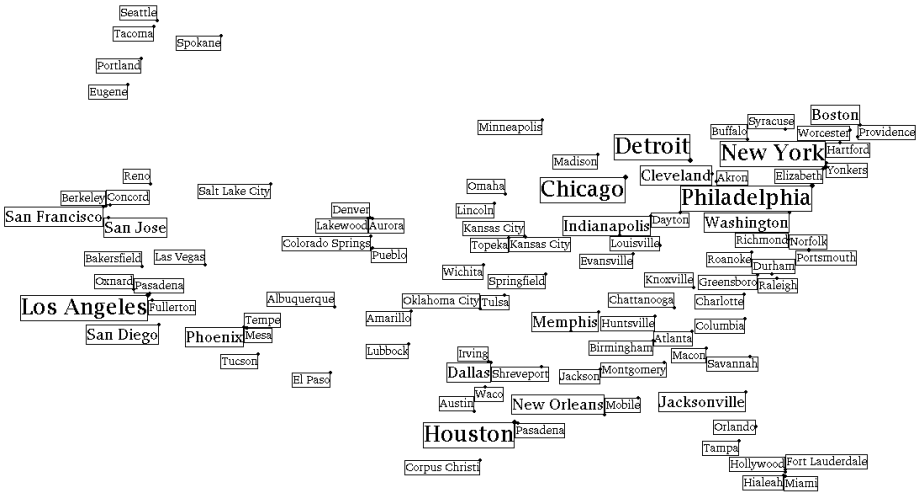

Figure 9: 156 cities in the U.S.A.; 97 labels placed (font-after-font method)

5 Conclusions

We have described two practical ways of extending recent research on point labeling in the sliding label model. The sliding label model allows labels to be placed anywhere as long as a side of the label touches the point to be labeled. This additional freedom in placement is known to allow more labels than the more widely known fixed position models [10].

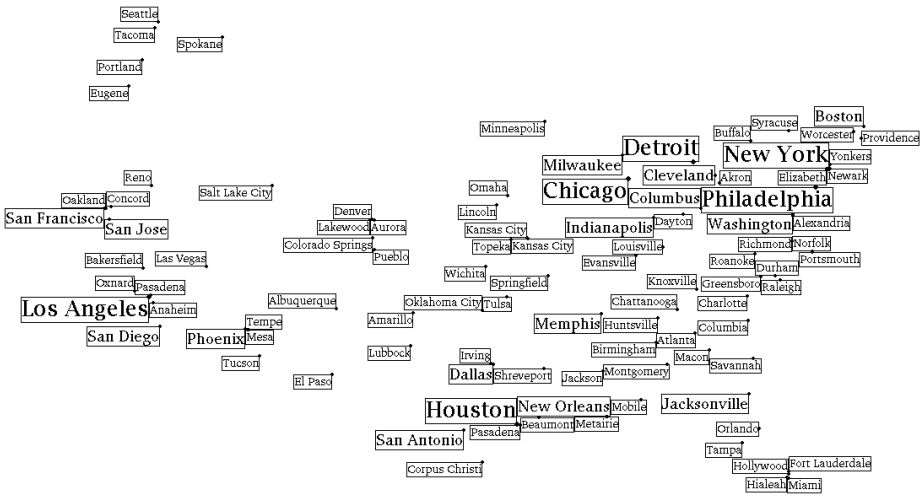

Figure 10: 156 cities in the U.S.A.; 100 labels placed (imaginary width method)

The first of the two extensions we have given deals with other map objects that must be avoided by the labels to be placed.

The second extension to sliding label position computation involved dealing with labels of different font sizes. We showed that the known sliding label algorithm can be adapted without much effort for this case. However, the algorithm will give preference to smaller labels over larger labels when it has the choice, and hence, to labels in smaller fonts. We presented approaches to correct this shortcoming, so that the more intuitive idea of giving preference to larger (more important) labels is used. We tested the font-after-font and the imaginary label width method on a map of the U.S.A. By giving cities with a large font a smaller imaginary label width we could place almost all of them.

The extension for avoiding other map objects runs in $O((n+m)\log(n+m))$ time with n the number of points to be labeled and m the combinatorial complexity of the map features that must be avoided. The algorithm which deals with labels of different font sizes runs in $O(n \log n)$ time. Furthermore, these extensions can be used together without any problem.

References

[1] Georges Alinhac. *Cartographie Théorique et Technique*, chapter IV. Institut Géographique National, Paris, 1962.

[2] Jon Christensen, Joe Marks, and Stuart Shieber. An empirical study of algorithms for point-feature label placement. *ACM Transactions on Graphics*, 14(3):203–232, 1995.

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

[4] Mark de Berg, Mark van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, second edition, 2000.

[5] Srinivas Doddi, Madhav V. Marathe, Andy Mirzaian, Bernard M.E. Moret, and Binhai Zhu. Map labeling and its generalizations. In *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms (SODA'97)*, pages 148–157, 1997.

[6] Michael Formann and Frank Wagner. A packing problem with applications to lettering of maps. In *Proc. 7th Annu. ACM Sympos. Comput. Geom. (SoCG'91)*, pages 281–288, 1991.

[7] Herbert Freeman. Computer name placement. In D.J. Maguire, M.F. Goodchild, and D.W. Rhind, editors, *Geographical Information Systems: Principles and Applications*, pages 445–456. Longman, London, 1991.

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

[9] Eduard Imhof. Positioning names on maps. *The American Cartographer*, 2(2):128–144, 1975.

[10] Marc van Kreveld, Tycho Strijk, and Alexander Wolff. Point labeling with sliding labels. *Computational Geometry: Theory and Applications*, 13:21–47, 1999.

[11] Frank Wagner and Alexander Wolff. A practical map labeling algorithm. *Computational Geometry: Theory and Applications*, 7:387–404, 1997.

[12] Alexander Wolff and Tycho Strijk. A map labeling bibliography. `http://www.math-inf.uni-greifswald.de/map-labeling/bibliography/`, 1996.

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