Point Labeling with Sliding Labels ⋆

Marc van Kreveld [a,2] Tycho Strijk [a,2] Alexander Wolff [b,1]

[a] *Department of Computer Science, Utrecht University, P.O. Box 80089, 3508 TB Utrecht, The Netherlands*

[b] *Institut für Informatik, Fachbereich Mathematik und Informatik, Freie Universität Berlin, Takustraße 9, 14195 Berlin, Germany*

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. We show that maximizing the number of labeled points is NP-hard in the most general of the new models. Nevertheless, we give a polynomial-time approximation scheme and a simple and efficient factor-$\frac{1}{2}$ approximation algorithm for each of the new models.

Finally, we give experimental results based on the factor-$\frac{1}{2}$ approximation algorithm to compare the models in practice. We also compare this algorithm experimentally to other algorithms suggested in the literature.

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, 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 [13], point sets are also common in data postings of field measurements, scatterplots of principal component analysis, and variograms, for instance. The ACM Computational Geometry

⋆ 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.

[1] awolff@inf.fu-berlin.de

[2] {marc,tycho}@cs.uu.nl

Impact Task Force report [3] denotes label placement as an important research area.

Generally it is assumed that a point label can be seen as an axis-parallel rectangle; the bounding box of the text. 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 [8,22]). The more general problem of map labeling includes line feature labeling (roads, rivers) and area labeling (countries) as well.

Good point labeling has two basic requirements. A label should be placed close to the point, to which it belongs, and two labels should not overlap each other. For high quality cartographic label placement, further requirements have been formulated [12,23]. Given the 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. In general, both of these problems are NP-hard [17,9].

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) [9,21,20], eight (many papers in the automated cartography literature), or any constant number [1]. 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. 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 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.

Slider models have been used in two previous papers. In Hirsch's paper [11], repelling forces are defined 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 is completely different from our approach, which is combinatorial. The paper by Doddi et al. [7] contains a number of labeling problems and algorithms, each using a different labeling model. One of the problems is solved in a slider model, where each label is allowed to rotate around the point to be labeled. The labels must be equal-size squares (or other regular polygons); the objective is to maximize the label *size*.

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.

Point labeling has long been shown to be NP-complete for fixed-position models [10,17,9,15]. However, this does not imply that label placement is also hard for slider models. In Section 3 we show that this is the case; we prove that it is NP-complete to decide whether a set of points can be labeled in the four-slider model.

In Section 4, 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 [1]. Our algorithm is greedy in that it always places the label whose right edge is leftmost among the right edges of all possible label placements. The algorithm uses a kind of generalized sweep-line in order to select the next label. We remark that our algorithm can be adapted to labels of varying height, but then the approximation factor does not hold any more.

In Section 5, we give a polynomial time approximation scheme for each of our slider models, showing that for any constant $\epsilon > 0$, there is a polynomial time algorithm 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 fixed-position but not for slider models.

In order to support the practical relevance of the greedy algorithm, we do a thorough experimental analysis in Section 6. We have implemented our greedy algorithm for the six models. We test it on three data sets from different application areas. One contains 1000 cities of the USA, 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 for a range of font sizes. It appears that the greedy algorithm produces about 10–15% more labels for a slider model than in the corresponding fixed-position model. This improvement is significant, because more labels are placed in dense areas. We also compare our algorithm to a simulated annealing algorithm proposed by Christensen et al. [4] on a sequence of randomly generated point sets.

2 Comparing Labeling Models

In this section we introduce and then compare some common point-labeling models. All of the algorithms we present in the following sections aim to label as many points as possible according to the chosen model.

Definition 1 (point labeling, size of a labeling, optimum labeling) *Given a set P of n points in the plane, and for each point $p \in P$ a set of*

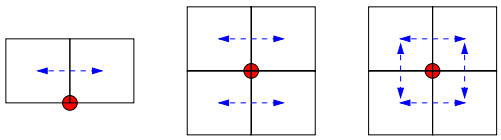


Fig. 1. top-, two- and four-slider model

label candidates L_p*, a* point labeling *is a subset* $P' \subseteq P$ *and a function* λ *which maps every point* $p \in P'$ *to a label* $\lambda(p) \in L_p$ *such that no two labels intersect. The number of labeled points, i.e. the cardinality of* P'*, is the* size of the labeling. *An* optimum labeling *labels a subset* $P' \subseteq P$ *of maximum cardinality. We denote the size of an optimal labeling by* k_{opt}.

In this paper, we restrict ourselves to axis-parallel rectangular label candidates. If we require additionally that a label must be placed such that one of its edges contains the point to be labeled, we get the following labeling models.

Definition 2 (slider models) *In the* four-slider model*, a point* p *must be labeled such that any edge of the label contains* p*. In the* two-slider model*, either the label's top or bottom edge has to contain* p*. In the* one-slider *or* top-slider *model, the bottom edge of a label must contain* p*.*

For an illustration of slider models, see Figure 1. Note that in all of our models we allow that a label contains other points which then of course cannot be labeled. Our labels are closed, i.e., we disallow touching. One alternative would be "half-open" labels as in [21]. In that paper all edges of a label which are not adjacent to its point are allowed to touch other labels or points. This would make sure that if every label is scaled down by a small amount with its point as scaling center, then all labels are disjoint. When labels do not touch, a map user can more easily match a label and the point to which it belongs. The algorithms could be adjusted to this additional requirement, but intensive case studying would be necessary to decide whether a label can be placed when it touches other labels. The bounds of the following comparison of models would still hold, but for the sake of simplicity we keep the number of requirements to a minimum.

One alternative would be to consider labels open and thus allow touching generally. In this case however, we were not able to keep the greedy algorithm's approximation guarantee of 50%, although the bounds of the comparison below would hold.

We will compare the slider models introduced above to the following fixed-position models.

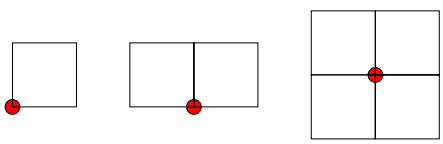


Fig. 2. one-, two- and four-position model

Definition 3 (fixed-position models) *Labeling in the* four-position model *requires that the a point p is labeled such that one of the label's corners lies on p. In the* two-position model *one of the label's bottom corners must lie on p and in the* one-position model *the lower left corner of the label must coincide with p.*

For an illustration of fixed-position models, see Figure 2. Our measure for comparing the models above is based on optimal labelings of point sets. Some point sets allow a labeling of the whole set in all models. Such point sets are not very interesting for a comparison, so we are mainly interested in point sets where the size of an optimal labeling differs from model to model. We define the *ratio* of two models as follows.

Definition 4 (ratio of two models) *Given unit square label candidates and two label-placement models M_1 and M_2, the (asymptotic)* $(M_1 : M_2)$-ratio *is*

$$\lim_{n\to\infty} \max_{P,\,|P|=n} \frac{\text{size}(\textit{optimal } M_1\textit{-labeling for } P)}{\text{size}(\textit{optimal } M_2\textit{-labeling for } P)}.$$

This measure does *not* take into account aesthetical criteria as listed by Imhof [12]. Since it is a purely quantitative measure and, moreover, only refers to square labels, it doesn't apply directly to many practical label placement problems. However, it gives a fair indication of how many more points can be labeled in one model than in another in general.

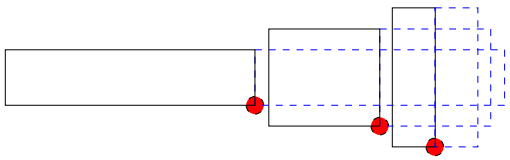


Fig. 3. The ratio between the two- and the one-position model can get arbitrarily bad for labels of different size.

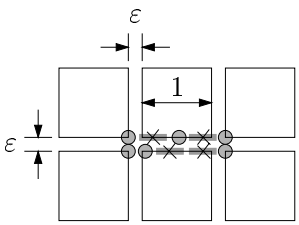


Fig. 4. 3/2 is a lower bound on the ratio between the two- or four-slider and any fixed-position model

The reason why we only consider unit square labels in the definition above and in the remainder of this section, is that otherwise some of the ratios between two models would become arbitrarily bad, see Figure 3. All points depicted there can be labeled in the two-position model, but only one point can be labeled in the one-position model.

It is worth mentioning that the size of an optimal placement in a slider model cannot be approximated arbitrarily well by a fixed-position model, no matter at how many discrete positions a fixed-position label can be attached to its point. Consider the six points marked by disks in Figure 4. The two leftmost points have distance ε from each other and distance $1 + 2\varepsilon$ from the corresponding rightmost points. These four points can be labeled in any model that allows a label's corner to lie on the point to be labeled. The cross markers

in the figure indicate the discrete positions at which labels for the other two points must be attached to their points in a given fixed-position model. If we choose ε smaller than half the minimum distance between any two such positions, then we can place the last two points such that they can be labeled in the two- or four-slider model, but not in the given fixed-position model. The grey bars of length 2ε centered at the cross markers in Figure 4 indicate the regions, in which a point *could* be labeled in the given fixed-position model. Repeating the group of six points yields a ratio of 3/2 between these models.

In this section a labeling model will always be one of those introduced in Definition 2 and 3. All of our comparisons of two such models M_1 and M_2 are based on 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 that is valid in the more restrictive model M_2. This might 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 of non-intersecting labels, remove l and all its conflicting labels from the instance and repeat until no labels remain. At the end of the process, S contains at least $k^1_{\text{opt}}/(\delta_{\text{left}}+1)$ non-intersecting M_2-labels, where k^1_{opt} 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. Lower bounds for the ratio ρ are obtained by giving examples of arbitrary size, for which any M_2-placement is worse by a certain factor than some M_1-placement.

Since we do not want to compare every two models in isolation, we define two groups. They consist of pairs of models where one model can be *flipped* and *slid* into the other, respectively.

Definition 5 (flipping) *Given two different label placement models M_1 and M_2, and an axis-parallel vector v of unit length, model M_1 can be* flipped into model M_2 by v *if any label position in M_1 that is not allowed in M_2 can be translated by v into a valid M_2-label position.*

Example 6 *The two-slider model can be flipped into the top-slider model by $(0,1)$. Analogously, the four-position model can be flipped by $(0,1)$ into the two-position model, while the two-position model can be flipped by $(1,0)$ into the one-position model.*

Lemma 7 *For any two labeling models M_1 and M_2, for which M_1 can be flipped into M_2, the $(M_1 : M_2)$-ratio is 2.*

PROOF. Consider an optimal M_1-labeling of an arbitrary instance of points. Let M_2 be a model into which M_1 can be flipped by a vector v. Then the canonical re-labeling mentioned above means translating by v all M_1-labels that are not valid in M_2.

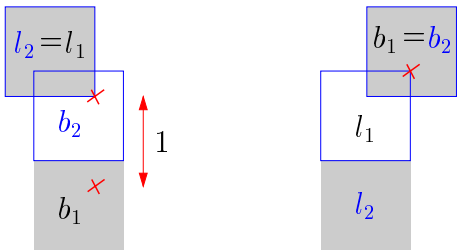


Fig. 5. If M_1 can be flipped into M_2 then the leftmost M_2-label l_2 (solid edges) cannot intersect more than one M_2-label b_2. (The corresponding non-intersecting M_1-labels are shaded.)

We can assume that the vector by which we flip is $(0, 1)$; the case $(1, 0)$ is symmetric. This means that an M_2-label is either identical to the corresponding M_1-label or lies one unit above it. Let l_1 be the M_1-label corresponding to the leftmost M_2-label l_2. We show that l_2 can intersect at most one M_2-label whose M_1-counterpart is not in conflict with l_1. As indicated above, this gives us an upper bound of 2 for the $(M_1 : M_2)$-ratio ρ.

Suppose that l_2 is identical to the corresponding M_1-label l_1; the other case is symmetric, see the left and right part of Figure 5, respectively. Let I_2 be the set of all M_2-labels intersecting l_2 and let I_1 be the set of their mutually non-intersecting M_1-counterparts. Then all labels in I_2 must contain the lower right corner of l_2; otherwise, either their M_1-counterparts intersect l_1, or l_2 is not leftmost. This however forces all labels in I_1 to contain a point at unit distance below that corner (marked by a cross in Figure 5) in order not to intersect l_1. Hence $|I_1| = |I_2| \leq 1$ and $\rho \leq 2$.

In order to establish the lower bound of 2 for ρ, just take the four corner points of an axis-parallel square of edge length less than one. For all models M_1 that we are considering and that can be flipped into a model M_2 (see Example 6), exactly twice as many of these points can be labeled as in the corresponding M_2-model. An instance can consist of arbitrarily many of such groups of four points, separated sufficiently. □

Definition 8 (sliding) *Given two different label placement models M_1 and M_2, and an axis-parallel vector v of unit length, model M_1 can be* slid into model M_2 along v *if every label position in M_1 can be translated by λv into a valid M_2-label position for some $\lambda \in [0, 1]$*

Example 9 *The four-slider model can be slid into both the two-slider and the top-slider model along $(0, 1)$. Along $(1, 0)$ we can slide the two-slider into the four-position model and the top-slider into both the two- and the one-position model. Note that the four-slider model cannot be slid into the four-position model.*

Lemma 10 *Let M_1 and M_2 be two (different) labeling models, and assume that M_1 can be slid into M_2. Then we have $2 \leq \rho \leq 3$ where ρ is the $(M_1 : M_2)$-ratio.*

PROOF. Again we consider an optimal M_1-labeling of an arbitrary instance. We assume that we can slide M_1- into M_2-label positions along $(0,1)$; the case $(1,0)$ is symmetric. We canonically slide all M_1-labels upwards until we 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.

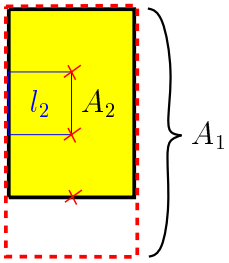


Fig. 6. If M_1 can be slid into M_2 then the leftmost M_2-label l_2 cannot intersect more than two M_2-labels.

M_2-labels intersecting l_2 can only lie within area A_2 in Figure 6 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 grid points in the interior of A_1 marked by crosses in Figure 6. 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 hence that $\rho \leq 3$.

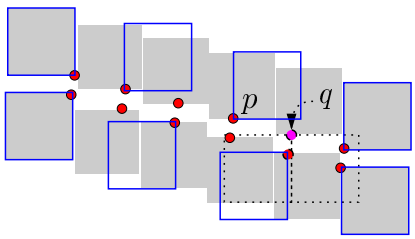


Fig. 7. If M_1 can be slid into M_2 then the $(M_1 : M_2)$-ratio is at least 2. Here we chose M_1 to be the two-slider model (shaded labels) and M_2 to be the four-position-model (solid edges).

For a lower bound of 2 refer to Figure 7. There are two rows of n points. Two neighboring points of one row have x-distance $1 - \frac{1}{n-1} + \varepsilon$ and y-distance δ, where $\frac{1}{n-1} > \delta > \varepsilon > 0$. The upper row is a copy of the lower, shifted by $(\delta/2, \delta)$.

Comparing the top-slider model to the one- or two-position model is easy; just consider one row. In order to compare the four- to the two-slider model, the figure must be rotated by 90 degrees. So let us focus on comparing the two-slider to the four-position model here.

It is obvious that all points can be labeled in the two-slider model. For the four-position model we can argue as follows. Whenever a four-position label is attached to a point which (like p) does not lie at the extreme left or right, then either the label contains at least one other point, or there is a point (like q)

in the vicinity whose label is then forced to contain at least two other points. □

Note that the proof above can be simplified for closed labels. We chose to give a proof that carries over to the case of open labels.

Lemma 11 *Let $\rho_{4S,4P}$ be the ratio between the four-slider and the four-position model. Then $2 \leq \rho_{4S,4P} \leq 4$*

PROOF. In order to show the upper bound for $\rho_{4S,4P}$, we assume an optimal four-slider labeling and canonically re-label the points as follows. We slide a label along its point either downwards or to the right until the point lies in one of the four corners. We use similar arguments as in the proof of Lemma 10 to show that the leftmost label l_{4P} in the resulting instance can intersect at most three other four-position labels. Such labels can only lie within area A_{4P} in Figure 8 since l_{4P} is leftmost. Let L_{4S} be the set of the corresponding non-

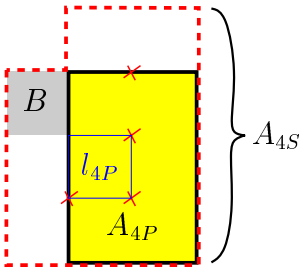


Fig. 8. The leftmost four-position model label l_{4P} cannot intersect more than three other four-position model labels.

intersecting four-slider labels. Labels in L_{4S} can lie at most one unit to the left or above their four-position counterparts. Thus they are restricted to area A_{4S} in Figure 8. Let l_{4S} be the four-slider counterpart of l_{4P}. Since l_{4S} was slid either down or to the right (or not at all), it follows that l_{4S} contains the upper left corner of l_{4P}. Consequently, no other label in L_{4S} can intersect the unit square region left and above the upper left corner of l_{4P}, denoted B in the figure.

The interior of $A_{4S} - B$ contains four grid points, and can therefore contain at most four non-intersecting labels of L_{4S} including l_{4S} itself. It follows that l_{4P} cannot intersect more than three four-position labels and that $\rho_{4S,4P} \leq 4$.

The lower bound of 2 is shown as in the proof of Lemma 10, see Figure 7. □

Lemma 12 *Let $\rho_{1S,1P}$ be the ratio between the top-slider and the one-position-model. Then $2\frac{1}{4} \leq \rho_{1S,1P} \leq 3$*

PROOF. With Lemma 10 we get $2 \leq \rho_{1S,1P} \leq 3$. The example in Figure 9 raises the lower bound to $2\frac{1}{4}$. □

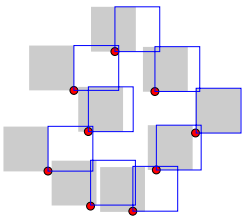


Fig. 9. Sliding top-slider labels (shaded) to the right into one-position labels (solid edges) can create 9-cycles in the resulting conflict graph of one-position labels.

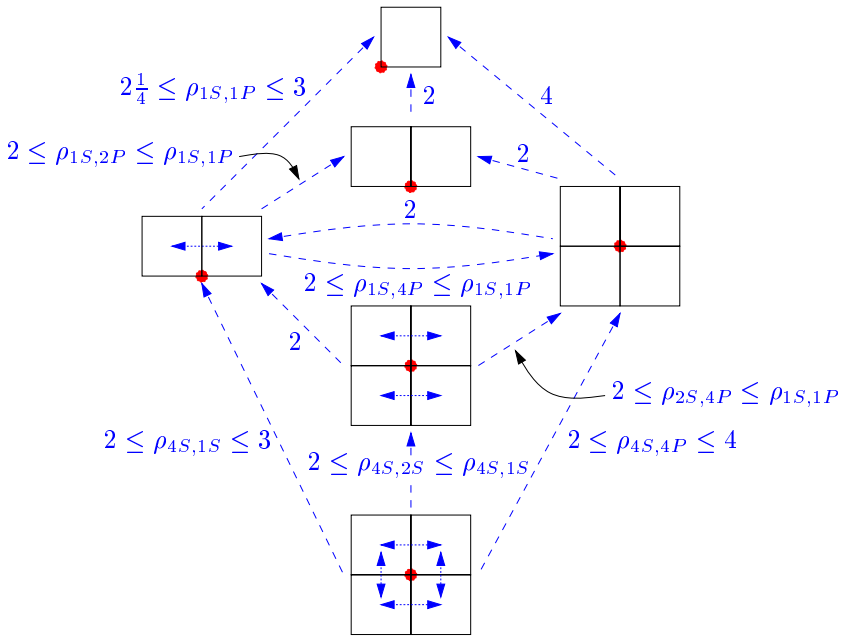


Fig. 10. Ratios between some label placement models. From top to bottom: the one-position, two-position, top-slider (left), four-position (right), two- and four-slider model

Figure 10 summarizes our results. In some cases we can replace an upper bound of 3 by the ratio between two other models like in the case of the top-slider model: The ratio $\rho_{1S,2P}$ between the top-slider and the two-position model is bounded from above by the ratio $\rho_{1S,1P}$ between the top-slider and the one-position model since the two-position is more general than the one-position model.

The reason for $\rho_{2S,4P} \le \rho_{1S,1P}$ is the following. We show $\rho_{2S,4P} \le \rho_{1S,2P}$ and then argue as above. In order to establish a relationship to the ratio between the one-slider and the two-position model, we take an arbitrary instance and put a copy of every point at unit distance below. (To avoid trouble with an original point at the same spot, we can move all copies upwards by an infinitesimal amount.) The optimal solutions in the one-slider and the two-position model on this instance correspond one-to-one to optimal two-slider and four-position solutions of the original instance, respectively. This trick is also used to generalize our algorithms in the following sections from top-slider

to two-slider labeling.

3 NP-completeness

The complexity of labeling points with axis-parallel rectangular labels from a *finite* set of label candidates is well established in the literature.

Independently, Marks and Shieber, and Formann and Wagner showed that it is NP-hard to decide whether a set of points can be labeled with squares in one of four positions [17,9]. The allowed positions are identical to those of our 4-position model. Marks and Shieber note that their proof also holds for models with an infinite number of positions like that of Hirsch [11], where a label must touch a circle of small radius centered on the point to be labeled. However, their proof cannot be used for our slider model. Fowler et al. [10], and Knuth and Raghunathan [15] have proved the NP-hardness of two other labeling models.

Recently, Iturriaga and Lubiw have proven that it is NP-hard to decide whether a set of points can be labeled with what they call left-right sliding labels [14]. Their model is similar to our 2-slider model except that they allow labels to touch both labels and sites of other labels. In their proof they use axis-parallel rectangular labels of varying size.

Slider models are a generalization of those fixed-position models that force a label to touch the point to be labeled. However, this observation does not yield the NP-hardness of the slider models, since it is not clear how an instance for a fixed-position model can be reduced to an instance of a slider model. Recall for example that the NP-completeness of 0-1-integer programs does not apply to their relaxation. Therefore we show that placing unit square labels in the 4-slider model is NP-complete.

Theorem 13 *It is NP-complete to decide whether a set of points can be labeled with axis-parallel unit squares in the 4-slider model.*

PROOF. The problem is in $\mathcal{NP}$ for the following reason. We can guess (i.e. compute non-deterministically) a permutation of the points and an integer between 1 and 4 for each point. This number indicates which edge of a label will be attached to the point. Then we go through the points according to the permutation and check for each point whether we can label it such that its label touches it on the chosen edge — given the labels we have already placed. If the new point can be labeled, we move its label into a *canonical* position: Depending on whether the pre-computed edge is horizontal or vertical, we slide the label along this edge as far left or down as possible. If all points can be labeled this way, we accept. Otherwise we discard the subset. The reason why we can reject in this case is the following. If all points could be labeled, we could push all labels in their canonical positions and name a permutations

of the points, such that the procedure outlined above would produce the same canonical label placement.

The proof of the NP-hardness is by reduction from planar 3-SAT. Lichtenstein showed that this restriction of 3-SAT is NP-hard [16]. Our proof follows Knuth and Raghunathan's proof of the NP-hardness of the metafont labeling problem [15]. We encode the variables and clauses of a boolean formula ϕ of planar 3-SAT type by a set of points such that all points can only be labeled if ϕ is satisfiable, i.e. if there is a variable assignment such that all clauses evaluate to true. The advantage of using a planar 3-SAT formula as opposed to the general type is the following. In planar 3-SAT, the variables can always be arranged on a straight line such that they are connected by *non-intersecting* three-legged clauses, see [15, Figure 5].

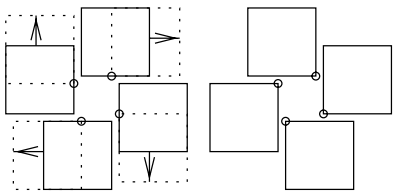


Fig. 11. Label placements encoding *true* and *false*.

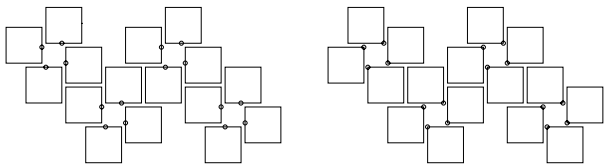


Fig. 12. Zig-zagging cluster patterns model a variable; the labels its Boolean value.

The main observation leading to our proof is the following. Given a cluster of four points (the corner points of a square with edges slightly longer than 1/2 and rotated by a small angle against the axes), there are two fundamentally different ways to label these points, see Figure 11. Under the condition that all points have to be labeled, the points can only be labeled as on the left side (which allows some sliding) or on the right side (where the labels are nearly fixed) of Figure 11. Note that is impossible that some points are labeled as on the left and others as on the right side. This gives us a means to encode the Boolean values of a variable in the planar 3-SAT formula ϕ that we want to reduce to a set of points.

The building blocks (or "gadgets") of our reduction are the clusters for variables, three-legged "combs" for clauses, and adapters connecting variables to clauses. In order to be able to connect a variable to all clauses in which it occurs, we model it not by one but by several four-point clusters in a zig-zag pattern as shown in Figure 12. Then still all points have to be labeled according to one of the two schemes mentioned above.

We model the clauses by point sets which resemble large combs with three legs, see Figure 13. The fourth column of points from the right and the left can be repeated as often as needed to reach the three variables belonging to the clause. The legs can be extended by duplicating the bottom-most row of points. Each leg is connected to a variable by an adapter. An adapter consists of three points a, b, and c. There are two types of adapters, see Figure 15 and 16. Which type is chosen depends on whether the variable is negated in

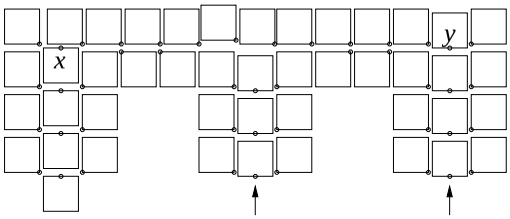


Fig. 13. Clause with pressure from two variables.

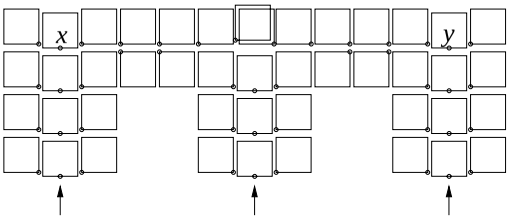


Fig. 14. Clause with pressure from three variables.

the clause. If the variable is set to a value which negates the corresponding literal in the clause, the lowest point b in the adapter must be labeled upwards, i.e. the label sticks into the pipe leading to the clause in question. This forces all other points above b to have their label above them as well. Graphically spoken, pressure is transmitted. This is indicated by arrows in Figure 13 to 16. When the pressure arrives in the top row of points in the representation of a clause, it is transmitted further horizontally, see the labels of the points x and y in Figure 13 and 14. Note that a variable assignment which fulfills the corresponding literal does not force anything; no pressure is exerted.

If all literals of a clause evaluate to false, then the points of type b in the adapters of the corresponding variables are labeled upwards and pressure is transmitted through all three vertical pipes into the clause. In this case there is a point which cannot be labeled, see Figure 14. If, however, there is at least one vertical pipe without pressure, all points belonging to a clause can be labeled, see Figure 13.

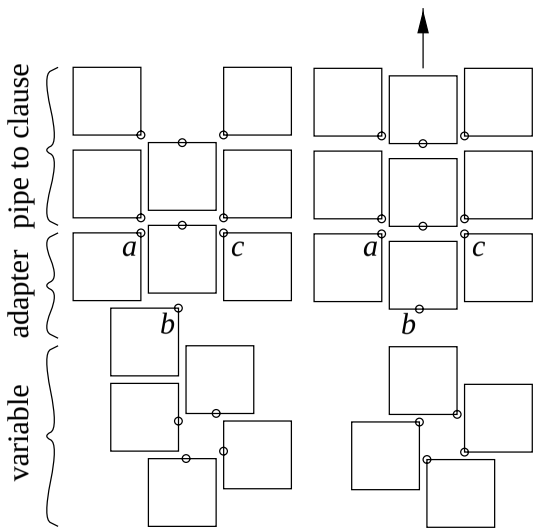


Fig. 15. Adapters for unnegated literals exert pressure when a variable is set to *false*.

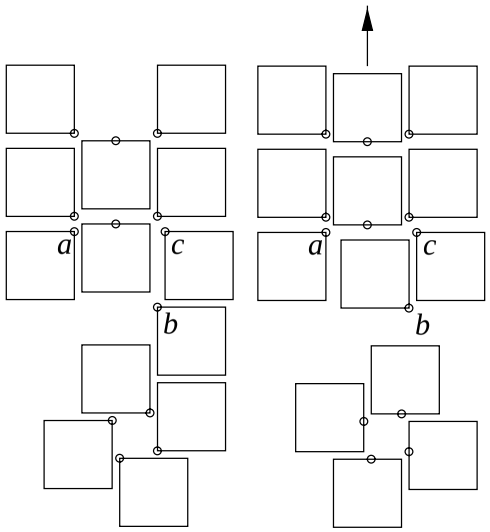


Fig. 16. Adapters for negated literals exert pressure when a variable is set to *true*.

Hence the question whether ϕ is satisfiable is equivalent to asking whether all points can be labeled in the 4-slider instance to which ϕ is reduced. It is easy to see that the reduction is polynomial: if ϕ consists of m clauses,

the instance has $O(m^2)$ points. Their position can certainly be computed in polynomial time. □

4 Greedy approximation algorithms

In this section we describe algorithms for point feature labeling in the slider models. They apply to labels of fixed height but arbitrary width. We describe an $O(n \log n)$ 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_{opt}, then our algorithm places at least $k_{\text{opt}}/2$ 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, a simple $O(n \log n)$ time, factor-$\frac{1}{2}$ approximation algorithm was described recently by Agarwal et al. [1]. We obtain the same result for the slider models. We'll only describe the most general four-slider algorithm; it is an extension of the top-slider and two-slider algorithms. It is based on a greedy strategy. For convenience we'll first describe the algorithm with labels allowed to touch, unlike in the previous sections where labels were considered to be closed. Later 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.

Lemma 14 *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.*

PROOF. Given a set P of points and a sliding model M, let L_{opt} be an optimum M-labeling. Let L_{left} be the set of labels computed by the greedy strategy. The set L_{left} is maximal in the sense that no label can be added to it without intersecting another label in L_{left}. So any label in L_{opt} must either be in L_{left} as well, or intersect some label in L_{left}, whose right edge is at least as much to the left. Charge each label in $L_{\text{opt}} - L_{\text{left}}$ to a label in L_{left} that lies as least as much to the left and intersects it. For any label in $L_{\text{opt}} \cap L_{\text{left}}$, charge it to itself.

We claim that any label in L_{left} is charged at most twice, from which the lemma follows. For labels in $L_{\text{opt}} \cap L_{\text{left}}$ the claim is obviously true. For any other label $l \in L_{\text{left}}$, observe that a label of L_{opt} that charges l must intersect the closed right edge of l. Since all labels have unit height, and the labels

in L_{opt} don't intersect each other, there can only be two labels of L_{opt} that intersect the closed right edge, and hence, charge l. □

A brute-force algorithm for this simple strategy would need $O(n^3)$ steps. In order to achieve an $O(n \log n)$ time bound, we must use some common geometric data structures.

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 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 still be accepted by the algorithm. Suppose a subset of the points has already received labels by the algorithm.

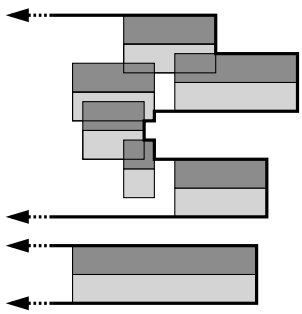


Fig. 17. Frontier of the placed labels (dark grey) and their lowered copies (light grey).

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 17. 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 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

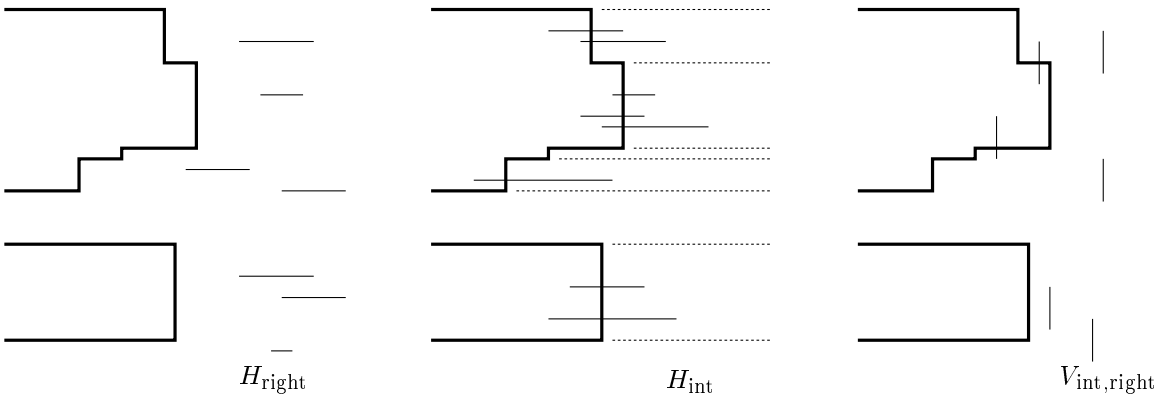


Fig. 18. The sets H_{right}, H_{int}, and $V_{\text{int,right}}$. The dashed lines in the middle picture separate the segments of H_{int} that are in different red-black trees $\mathcal{T}_i$.

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 18. 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 $\mathcal{P}$ [18]. These are also described in standard textbooks on algorithms [5] and computational geometry [6].

4.1 Leftmost label query structures

We use three data structures to find the leftmost label position among the ones represented by H_{int}, H_{right}, and $V_{\text{int,right}}$. They are:

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 $\mathcal{H}_{\text{right}}$, 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 18). 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 use a heap $\mathcal{H}_{\text{int}}$ to have fast access to the segment in H_{int} that allows the leftmost label placement. $\mathcal{H}_{\text{int}}$ stores for each $\mathcal{T}_i$ the sum of the x-coordinate of f_i and the minimum width of the segments in $\mathcal{T}_i$. Thus the root of $\mathcal{H}_{\text{int}}$ corresponds to the leftmost label among the labels represented by H_{int}.

3. For the vertical segments in V, we don't maintain the set $V_{\text{int,right}}$ 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 $\mathcal{H}_V$. 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 $\mathcal{H}_V$, we test whether 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.

4.2 Update assistance structures

After the leftmost label has been determined, 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. Let f_{new} be the right edge of the extended rectangle $\tilde{l}$ of the newly placed label l. The new frontier F is the right envelope of the old frontier and f_{new}, see Figure 19.

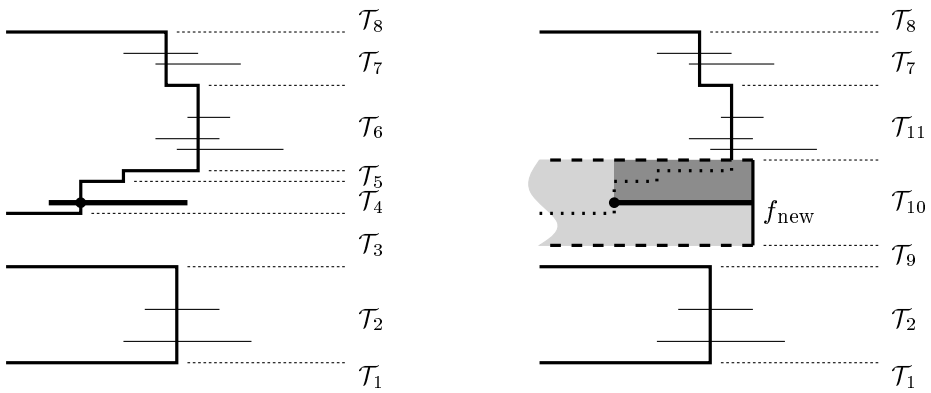


Fig. 19. When the fat horizontal segment s from H_{int} is chosen, the frontier becomes the right envelope of f_{new} and the old frontier. The new label is dark grey. The grey range (light and dark) is the one with which queries in the priority search trees are done.

1a. To determine which segments move from H_{right} to H_{int} or H_{left} when the frontier changes, we use a priority search tree $\mathcal{P}_{\text{left}}$ on the left endpoints of segments in H_{right}. After placing a label, we query $\mathcal{P}_{\text{left}}$ with the region left of f_{new} (grey in Figure 19) to locate the left endpoints of all segments

that are no longer in H_{right}. We delete these endpoints from $\mathcal{P}_{\text{left}}$, and we delete the corresponding segments from the heap $\mathcal{H}_{\text{right}}$. For each deleted segment we test whether its 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 move from H_{int} to H_{left} when the frontier changes, we use a priority search tree $\mathcal{P}_{\text{right}}$ on the right endpoints of segments in H_{int}. After placing a label, we query $\mathcal{P}_{\text{right}}$ with the region left of f_{new} (grey in Figure 19) to locate all right endpoints of segments that have moved from H_{int} to H_{left}. Then we delete the entries corresponding to these segments from the trees $\mathcal{T}_i$, from the heap $\mathcal{H}_{\text{int}}$ and from $\mathcal{P}_{\text{right}}$ itself.

When the frontier changes, we must also reorganize the red-black trees and $\mathcal{H}_{\text{int}}$ as a whole. Recall that we use a red-black tree $\mathcal{T}_i$ for each vertical segment of F. At most three new vertical segments can arise when the frontier changes, but many more vertical segments may cease to exist. We use the 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 19 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 by inserting the value of each new tree.

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

4.3 The algorithm

While there are still segments in any of the heaps $\mathcal{H}_{\text{int}}$, $\mathcal{H}_{\text{right}}$, or $\mathcal{H}_V$, do the following steps:

(1) Let v be the vertical segment that corresponds to the minimum of $\mathcal{H}_V$. Search with v in the augmented red-black tree $\mathcal{T}_V$ to see if v has some point right of F. If not, remove v from $\mathcal{H}_V$ and repeat this step.

(2) Determine the smallest among the minima of the three heaps $\mathcal{H}_{\text{int}}$, $\mathcal{H}_{\text{right}}$, and $\mathcal{H}_V$. Remove this minimum from its 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.

(3) Determine f_{new}, the right edge of the extended rectangle $\tilde{l}_i$. Update the

frontier F with f_{new}. Update the augmented red-black tree $\mathcal{T}_V$ (from **3a.**) with f_{new}.

Search with the region horizontally left of f_{new} (grey in Figure 19) in the priority search trees $\mathcal{P}_{\text{left}}$ and $\mathcal{P}_{\text{right}}$ (from **1a** and **2a**) and update the structures $\mathcal{H}_{\text{right}}$ (from **1**), $\mathcal{P}_{\text{left}}$(from **1a**), $\mathcal{H}_{\text{int}}$ and $\mathcal{T}_i$'s (from **2**), and $\mathcal{P}_{\text{right}}$ (from **2a**) as explained in the description of these structures.

(4) Remove all other reference segments corresponding to p_i from the data structures, in which they occur.

4.4 The 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 algorithm's runtime of $O(n \log n)$ follows from the following observations. Any vertical segment f_{new} creates one vertical edge in the frontier F, and shortens 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 in the whole algorithm (although many can be destroyed when one vertical segment f_{new} is added to the frontier). This bounds the total number of red-black trees $\mathcal{T}_i$ (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 $\mathcal{T}_V$ (from **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 $\mathcal{P}_{\text{left}}$ and $\mathcal{P}_{\text{right}}$. 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 initializing, querying and updating the priority search trees.

So far we have only discussed the placement of labels that were allowed to touch at the boundaries, that is, the disjoint placement of open rectangles. How can the ideas be adapted to incorporate closed rectangles as labels? Firstly, we let the frontier represent a closed region where reference points of labels

cannot lie any more. But the real problem is that we cannot choose and place the *leftmost* label, because this is not well-defined in the slider model with closed rectangles. The solution is to make a distinction between a placement of a rectangle at some position with x-coordinate $\bar{x}$ and a placement at some position with x-coordinate arbitrarily close to $\bar{x}$, but still strictly to the right of it. Such a distinction can be made by using a symbolic value $\epsilon > 0$ that is arbitrarily close to 0. In case of ties in x-coordinates of labels in the heap, one of them may have been moved symbolically to the right, which resolves the tie. If neither or both labels have been moved symbolically, there is a real tie and we can choose either label as the leftmost. When the algorithm finishes and a set of labels has been selected, then the actual positions of these label can be computed.

We conclude:

Theorem 15 *Given n points in the plane, and for each point a rectangular label with fixed height and some given width. Then for each of the fixed-position and slider models, there is an $O(n \log n)$ time and linear space algorithm which places at least half the maximum number of labels.*

Remark 16 *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 label positions. In the heap, we store the sum of x-coordinate and label width for each reference point. When the label corresponding to the heap's minimum is chosen, we query in the priority search tree with the appropriate range to find the reference points that are no longer valid. We remove the entries of these reference points from heap and priority search tree, and repeat by selecting the minimum from the heap.*

5 A Polynomial Time Approximation Scheme

In this section we present schemes for approximating the number of points we can label with unit height labels in all slider models. First we will only consider the top-slider model and then show how these results can be generalized to polynomial time approximation schemes for the two- and four-slider model.

5.1 Top-slider model

Given a constant $\varepsilon \in (0, 1)$ we show that there is an algorithm that finds a top-slider labeling of at least $(1 - \varepsilon) \cdot k_{\text{opt}}$ points, where k_{opt} is the number of labeled points in an optimal top-slider solution. The algorithm has running time $O(n^{4/\varepsilon^2})$.

We use line stabbing to split the problem into smaller units as suggested

in [1]. We stab the unit height labels with horizontal lines of distance strictly greater 1 such that each label is stabbed by exactly one line. This can be done in $O(n \log n)$ time [1] and gives us a partition of the set of input points P into disjoint sets $P_1, \ldots, P_m$, where P_i contains all points whose label intersects the i-th line, and m is the number of stabbing lines.

If we want to obtain an approximation ratio better than 1/2, we cannot afford to discard every second subset P_i of input points. Instead, we have to look at groups of t consecutive subsets. For $1 \leq i \leq t+1$, let

$$P^i = P - \bigcup_{j=0}^{\lfloor \frac{m-i}{t+1} \rfloor} P_{i+j\cdot(t+1)}$$

be the set of points that we get from P if we discard every $(t+1)$-st subset starting with P_i. This makes sure that if we compute the optimal solution for t consecutive lines, then we get an approximation for P^i since solutions for its blocks of t lines do not interfere with each other. The pigeon hole principle guarantees that one of the $t+1$ sets of type P^i has an optimal solution of size at least $\frac{t}{t+1} \cdot k_{\text{opt}}$. In [1] this approach was taken, where the optimal solution for the t-lines problem was solved by dynamic programming. In the case of sliding labels one cannot take this approach because the number of candidate label positions in the discretization is superpolynomial. We will still arrive at a polynomial time approximation scheme for the original problem by *approximating* the t-lines subproblem.

Suppose we find a $\frac{k}{k+t-1}$-approximation for the t-line problem, then we can approximate the original problem by a factor of $\gamma = \frac{k}{k+t-1} \cdot \frac{t}{t+1}$, which depends on the two parameters t and k. Setting $k = (t+1)(t-1)$ and $t = \lceil 2/\varepsilon \rceil - 2$ then yields $\gamma = t/(t+2) \geq 1 - \varepsilon$, the desired approximation factor. If the instance needs less than $\lceil 2/\varepsilon \rceil - 2$ stabbing lines, the solution of the problem becomes easier. In this case we set $k = (m-1)(\lceil 1/\varepsilon \rceil - 1)$ and approximate the m-line problem directly with a factor of $\gamma = \frac{k}{k+m-1} \geq 1 - \varepsilon$. The running time would then slightly improve to n^{k+1}. So we can assume $t \leq m$ from now on.

It remains to show how we can approximate an optimal solution for t lines by a factor of $\frac{k}{k+t-1}$. The idea is simple and uses the geometrical flavor of the problem. We call a labeling of a set of points *canonical* if all points are labeled and, going through the points from left to right, all labels have been pushed as far left as possible, that is, until they nearly hit another label or have arrived in their leftmost position. (Recall that labels are not allowed to touch each other. As in Section 4 we treat the distinction between an x-coordinate and a position slightly more to the right symbolically.) Now we just look at *all* canonical label placements of k points. For each such placement we consider the vertical line that goes through the right edge of its rightmost label. We search for the canonical labeling of k points with the leftmost such line ℓ_{left}, see

Figure 20. (We have *not* visibly drawn the infinitesimally small spaces between the labels.) We call this placement *leftmost* and compare it to the leftmost k

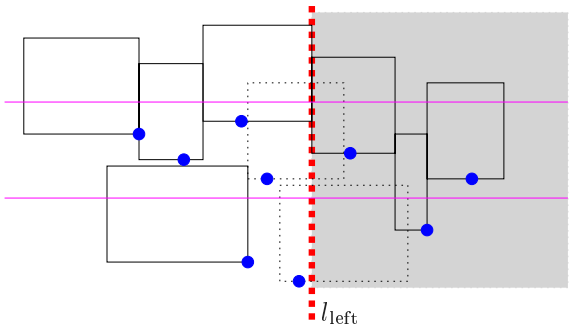


Fig. 20. Leftmost label placement for a subset of $k = 4$ points and $t = 2$ lines.

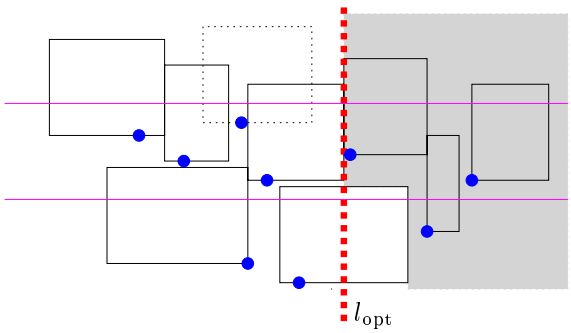


Fig. 21. An optimal solution for the same points as in the figure above.

labels of the optimum. Let ℓ_{opt} be the vertical line that goes through the k-th leftmost right label edge of the optimal solution, see Figure 21. Then we know that ℓ_{left} is at least as far to the left as ℓ_{opt}. We would like to repeat this process with all sets of k points to the right of ℓ_{left}. We must label them under the restriction that their labels can only be placed to the right of ℓ_{left}. If we do so, by how much do we get worse than the optimal solution?

By definition ℓ_{opt} touches one label of the optimal solution and intersects up to $t-1$ labels on the other $t-1$ lines. Since ℓ_{left} is not to the right of ℓ_{opt}, the constraint that our leftmost labeling exerts on the next group of k labels is no stronger than the constraint defined by the labels of the optimal solution touching or intersecting ℓ_{opt}, see the gray zones in Figure 20 and 21. Thus we have placed our first k labels in at most as much 'space' as the first $k+t-1$ labels of the optimal solution. This makes sure that the next line like ℓ_{left}, defined by the next (restricted) leftmost labeling of k points, will again be at most as far to the right as the vertical line through the $(2k+t-1)$-st leftmost right label edge of the optimal solution. By repeating this process until all points are used up, we get a $k/(k+t-1)$-approximation for the number of labeled points in an optimal solution since we always fit k labels in at most as much space as $k+t-1$ labels of the optimal solution. This shows that for the appropriate choice of t and k, we obtain a $(1-\varepsilon)$-approximation for the

whole problem.

Let n' be the number of points whose labels intersect a fixed set of t consecutive lines. What is the time we need to compute the first leftmost placement of k out of these n' points? We enumerate all $\binom{n'}{k}$ choices of these k points. For each choice we have to find its canonical labeling—if there is any. Observe that labeling a point p_1 can constrain the labeling of a point p_2 to its left only by not at all allowing to label it. Since we are only interested in subsets of k points that can be labeled completely, it is enough to go through the points in lexicographical order and try to place each of them leftmost. We can find a label's leftmost position by going through the list of its predecessors once, so finding a canonical labeling can certainly be done in $O(k^2)$ steps. This means that it takes us $O((n')^k)$ steps to compute the first leftmost labeling. Thus we need $T_{t-\text{line}}(n') = \sum_{j=0}^{\lceil n'/k \rceil} O((n' - jk)^k) = O((n')^{k+1})$ time for an approximate solution of the t-line problem. In order to get the total running time T_{total}, we must sum up $T_{t-\text{line}}$ over all possible groups of t consecutive lines. In every group there are at most n points and m, the number of stabbing lines, is at most n as well. Hence $T_{\text{total}}(n) = O(n^{k+2})$. Using $k = (t+1)(t-1)$ and $t = 2/\varepsilon - 2$ as above yields $T_{\text{total}}(n) = O(n^{4/\varepsilon^2})$.

5.2 Two and four sliders

This scheme for the top-slider model immediately translates into a polynomial time approximation scheme for the two-slider model. As we have seen in Section 2, for each point of the input set, we just have to place a copy at unit distance below it. Then only one point of every such pair is labeled in the solution. The running time increases only by a constant factor.

In order to use the ideas given above for the four-slider model, we have to do a little more work. Since labels can now move up and down, the use of stabbing lines is not appropriate any more. Instead, we partition the set of input points into m *strips* of unit height. A strip contains all points between its two bounding horizontal lines and all points that lie on the upper boundary. Similar to our approach above, we will approximate the solution of t consecutive strips. This time, however, we have to drop the points of *two* strips between two blocks to guarantee that solutions of one block do not interfere with solutions of an adjacent block. The pigeon hole principle makes sure that one of the $t+2$ different sets we get by gluing blocks together has at least cardinality $n \cdot \frac{t}{t+2}$. Suppose we have a $\frac{k}{k+t}$-approximation for the t-strip problem, then we could approximate an optimal solution of the whole instance by a factor of $\gamma = \frac{k}{k+t} \cdot \frac{t}{t+2}$. Setting $k = t(t+2)$ and $t = \lceil 3/\varepsilon \rceil - 3$ would then result in $\gamma = t/(t+3) \geq 1 - \varepsilon$, the desired approximation factor.

The additional difficulty in designing an approximation for the t-strip problem is that we do not know on which of its four sides a label in the optimal solution is attached to its point. We can handle this by considering all four

possibilities for each of the k points we have chosen. Now we define a canonical labeling as follows. If a label is to be attached to its point on the top or bottom edge, we again push it as far left as possible. If however its point is going to lie on the right or left edge, we push the label as far down as possible. The idea with considering a special order of the points does not work in this setting, so we try to label the k points in every of the $k!$ possible orders, and for every order we check each of the 4^k possible kinds of placement: left, right, bottom, or top. In this way we can again find a leftmost labeling and a line ℓ_{left}. The constraint that the leftmost labeling exerts on the next group of k labels is at most as strong as the corresponding constraint of the assumed optimal solution. As above, the constraint of the optimal solution is defined by ℓ_{opt} and the labels of the optimal placement intersected by ℓ_{opt}. Apart from the label whose right edge defines ℓ_{opt}, at most t labels can intersect ℓ_{opt} without intersecting each other since their points have to lie within a vertical strip of height strictly less than t (the bottom borderline is excluded). Hence we have a $\frac{k}{k+t}$-approximation for the t-strip problem.

In the approximation algorithm for the four-slider model, we need $\binom{n'}{k} k! 4^k k^2$ steps to compute the first leftmost labeling. This still yields an overall running time of $O(n^{4/\varepsilon^2})$.

Theorem 17 *For each of the slider models and for any constant $\varepsilon > 0$, there is a polynomial time algorithm which labels at least $(1-\epsilon)$ times the maximum number of input points that can be labeled.*

6 Implementation and test results

The greedy algorithm of Section 4 has been implemented for the fixed-position and slider models and tested on three real world data sets from different application areas and on a large sequence of randomly generated point sets. In this section we compare experimentally how many labels are placed in each of the six models.

The algorithms were implemented in C++. For the data structures we made use of the LEDA library [19]. We simplified the implementation described in Section 4 in three respects. Firstly, the red-black trees $\mathcal{T}_k$ can be expected to contain only a few horizontal segments at any moment. So we used simple lists for them. Secondly, LEDA doesn't have an implementation for priority search trees; we used orthogonal range trees instead. Thirdly, the augmented red-black tree $\mathcal{H}_V$ doesn't profit much from the augmentation in practice. When searching for the minimum x-coordinate of the frontier 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 USA that must be labeled with their name. We used several different font sizes, and determined

the bounding boxes of the label text. 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 corresponding slider models. The values in the second table show the results in percentages with respect to the 4-position labeling.

No. of labels placed

	model					
font	1P	2P	4P	1S	2S	4S
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

Percentage w.r.t. 4-position model

	model					
font	1P	2P	4P	1S	2S	4S
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
One thousand cities on a large map.

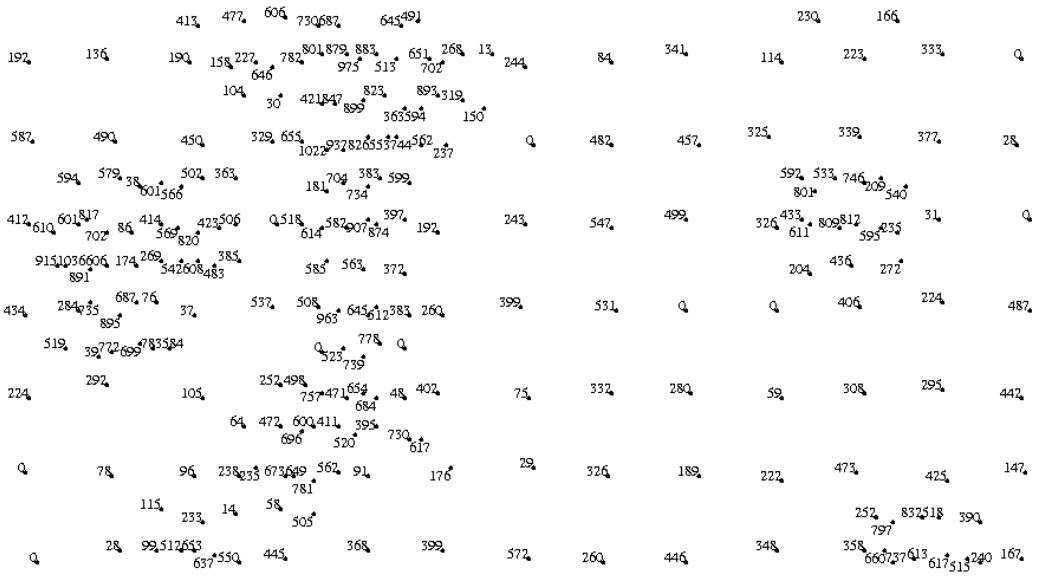
The second data set contains the 236 points of a data posting. The labels are measurement values and come from a book on geostatistics [13]. Figure 22 shows the labeled data set and the number of labels placed in each model.

The third data set contains 75 points of a regression analysis. Here the points are clustered near a regression line, and the labels are simply identification numbers. Figure 23 shows the labeling.

The bottom tables of Figures 22 and 23 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 of the areas that are difficult to label. We also created artificial, pseudo-random data sets where all areas are hard to label. These sets were constructed by first placing all points on a grid and after that they were moved randomly a slight distance away from the gridpoint. 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 seldom contains more than 1000 labeled points.

Christensen, Marks and Shieber compared different algorithms using random point sets [4]. Their standard data sets were generated as follows. Inside a grid of size 792 by 612 units, n points were randomly placed and had to be labeled with labels of 30 by 7 units. We considered examples with $n = 100, 250, 500, 750, 1000$, and 1500 points. For each example size, we gen-



No. of labels placed

font	model					
	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	model					
	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

Fig. 22. Labeling of the data posting in 9pt font using the 4-slider model (scaled to fit), and tables with the performance.

erated 25 files. We ran the greedy algorithm for each of our six models on all of the generated files. The average percentages of placed labels over the 25 trials is shown in Table 2. Clearly the labeling model has a big influence on the results.

In Figure 24 we extend the comparison of Christensen et al. by the results of our algorithm for the four-position and the four-slider model. Our four-position algorithm is always better than gradient descent, and the denser the map the better it gets in relation to gradient descent. For 1500 points it is almost as good as simulated annealing. The four-slider algorithm yields almost equal results as simulated annealing for less than 750 points and is always better

No. of labels placed

font	model					
	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	model					
	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

Fig. 23. Labeling of the scatterplot in 11pt font using the 4-slider model (scaled to fit), and tables with the performance.

beyond 750 points. The running time of our algorithm is generally only a few seconds; even the four-slider algorithm needed just 12 seconds for the largest data sets with 1500 points on a SUN Ultra Sparc. Simulated annealing takes several minutes to label these point sets on the same machine.

	Percentage of placed labels					
	number of points					
model	100	250	500	750	1000	1500
1P	92.60	84.30	73.16	64.56	57.96	48.58
2P	99.56	97.39	90.24	82.22	74.73	62.75
4P	99.84	99.07	95.45	90.47	83.99	71.74
1S	99.72	98.42	93.80	87.80	81.92	71.04
2S	99.92	99.55	97.83	94.85	90.71	80.75
4S	99.96	99.58	98.02	95.37	91.68	82.68

Table 2
Random data sets (results are averaged over twenty-five trials).

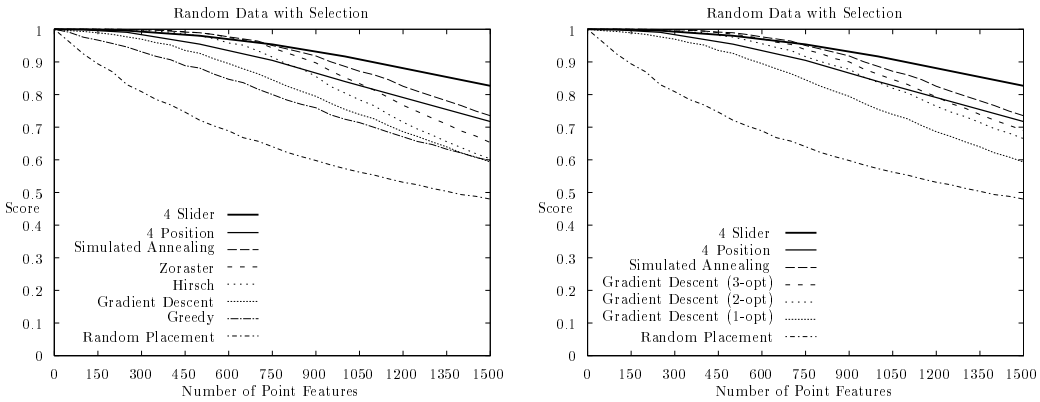


Fig. 24. Comparison of the four-position and four-slider algorithm to other labeling algorithms.

7 Conclusions and open problems

This paper has extended several existing results on point set labeling by allowing that the label touches its point anywhere on its boundary, not just at a finite set of positions. We have shown that removing this unnatural assumption can lead to labelings where more points receive a label than with the assumption, both in theory and in practice. We showed that simple and efficient greedy approximation algorithms, and polynomial time approximation schemes can still be developed in these more general labeling problems.

In our paper we only attempted to optimize the number of points that receive a label. The most important extension is to include other aspects of good map labelings in the algorithms as well, like avoiding ambiguity. Another extension is to deal with labels of varying height, and analyzing whether our

results and algorithms extend to this case as well. Finally, cartographic labeling requires that linear features like rivers, and area features like countries, receive a well-positioned label as well. This leads to a number of interesting issues for further research.

References

[1] Pankaj Agarwal, Marc van Kreveld, and Subhash Suri. Label placement by maximum independent set in rectangles. In *Proceedings of the 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] Bernard Chazelle et al. Application challenges to computational geometry: CG impact task force report. Technical Report TR-521-96, Princeton Univ., April 1996.

[4] 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.

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

[6] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, 1997.

[7] 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*, pages 148–157, 1997.

[8] Jeffrey S. Doerschler and Herbert Freeman. A rule-based system for dense-map name placement. *Communications of the ACM*, 35:68–79, 1992.

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

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

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

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

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

[14] Claudia Iturriaga and Anna Lubiw. NP-hardness of some map labeling problems. Technical Report CS-97-18, University of Waterloo, Canada, 1997.

[15] Donald E. Knuth and Arvind Raghunathan. The problem of compatible representatives. *SIAM J. Discr. Math.*, 5(3):422–427, 1992.

[16] D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982.

[17] Joe Marks and Stuart Shieber. The computational complexity of cartographic label placement. Technical report, Harvard CS, 1991.

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

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

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

[22] Alexander Wolff and Tycho Strijk. A map labeling bibliography. `http://www.inf.fu-berlin.de/map-labeling/bibliography/`, 1996.

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