International Journal of
COMPUTATIONAL GEOMETRY & APPLICATIONS

Volume 12 • Number 6 • December 2002

Labeling Points with Rectangles of Various Shapes

A. Koike, S.-I. Nakano, T. Nishizeki, T. Tokuyama & S. Watanabe

World Scientific
New Jersey • London • Singapore • Hong Kong • Bangalore

International Journal of Computational Geometry & Applications
Vol. 12, No. 6 (2002) 511–528

LABELING POINTS WITH RECTANGLES OF VARIOUS SHAPES

ATSUSHI KOIKE
Graduate School of Information Sciences
Tohoku University, Aobayama, Sendai 980-8579, Japan
koike@dais.is.tohoku.ac.jp

SHIN-ICHI NAKANO
Department of Information Engineering, Gunma University
Kiryu 376-8515, Japan
nakano@cs.gunma-u.ac.jp

TAKAO NISHIZEKI*, TAKESHI TOKUYAMA† and SHUHEI WATANABE‡
Graduate School of Information Sciences, Tohoku University,
Aobayama, Sendai 980-8579, Japan
**nishi@nishizeki.ecei.tohoku.ac.jp*
†*tokuyama@dais.is.tohoku.ac.jp*
‡*nabe@nishizeki.ecei.tohoku.ac.jp*

Received 9 November 2001
Revised 23 July 2002
Communicated by Mark de Berg

ABSTRACT

We deal with a map-labeling problem, named LOFL (Left-part Ordered Flexible Labeling), to label a set of points in a plane in the presence of polygonal obstacles. The label for each point is selected from a set of rectangles with various shapes satisfying the *left-part ordered* property, and is placed near to the point after scaled by a scaling factor σ which is common to all points. In this paper, we give an optimal $O((n+m)\log(n+m))$ algorithm to decide the feasibility of LOFL for a fixed scaling factor σ, and an $O((n+m)\log^2(n+m))$ time algorithm to find the largest feasible scaling factor σ, where n is the number of points and m is the total number of edges of the polygonal obstacles.

Keywords: Map labelling; geographic information systems; plane sweep algorithm; parametric search.

1. Introduction

Annotating a set of points is a common task to be performed in Geographic Information Systems. It is crucial that important objects in a map have labels indicating their names or other attributes. The objects to be labeled in a map highly depend on the user's interest; for example, a drainage maintainer may want to have locations and identification labels of manholes, although they are almost useless information for ordinary users. Therefore, a digital map should have access to a database of sets

of points representing locations of objects together with labels of the objects and a procedure that efficiently label any subset of the points in the database on demand.

The problem of placeing labels in a map is called the map labeling (or map lettering) problem.[14,23,24,25] Approximating a label (a string of characters) by its bounding rectangle, one can formulate the map-labeling problem as the problem of placing a set of n rectangles in a plane (with obstacles containing m edges) in a way that (1) each rectangle representing a label of an object should be near the object, (2) rectangles do not overlap each other, and (3) each rectangle does not overlap any obstacle in the map. Condition (1) will be formulated mathematically in a suitable fashion.

We restrict ourselves to the *point feature label placement problem*, where each object is a point (object point) in the map. The rightmost point of an object is a typical choice for an object point. Moreover, we only consider axis parallel rectangles as labels. See Refs. [3, 12] for more general labeling problems.

If the size of each character is given (therefore, the size of each label is given), we want to decide whether there exists a feasible solution satisfying conditions (1), (2), and (3) above. Such a problem is called *decision problem*. We also want to consider an *optimization problem* in which we compute the maximum character size σ, called *scaling factor*, for which there is a feasible solution.

The decision problem is hard in general: Formann and Wagner[8] showed that if each point has four label candidates, it is NP-hard in general to decide the feasibility. Indeed, it is NP-hard even if each label is a unit square and must be placed in a way that the corresponding object point is at one of its four corners (four-position model); we say that such a label is *pinned* at a corner. Kato[13] showed that the problem remains NP-hard if each point has three label candidates. Thus, we need to give stronger restriction to the labeling schemes in order to solve the labeling problem in polynomial time.

On the other hand, if each label is a unit square pinned at one of its two left corners (two-position model), the problem is polynomial time solvable. In general, if there are at most two candidates of the placement for each label, the problem is polynomial-time solvable since it can be formulated as a 2-SAT problem.[8] Moreover, approximation algorithms with provable approximation ratios are given for several useful versions of the map labeling problem.[23]

If we fix the scaling factor and measure the quality of the solution by the number of labels that can be placed without overlapping, there are PTAS algorithms for several cases,[1,14] and theoretical comparison between several map-labeling models is given.[14] There are several other labeling problems that can be solved in polynomial time.[22,15,19,9,7]

We will deal with another type of point-labeling problem, called *shape-flexible labeling*, where we can flexibly choose the shape of each label from a set of candidate rectangles. The chosen labels are placed after they have been scaled by a scaling factor σ which is the same for all labels. The problem of deciding the feasibility of a shape-flexible labeling problem is NP-hard in general, and the complexity of solving

the problem heavily depends on features of the candidate sets. If each candidate set is the set of all rectangles with a given area, the labeling is called *elastic labeling*, and some special cases were investigated by Iturriaga and Lubiw.[11,10]

Our motivation is as follows: Consider a rectangular label representing a character string of length l. It needs width l (character units) if it is written in a single line. However, we can fold the label to decrease the width. Moreover, in the Chinese (also Japanese or Korean) language system, we can write a character string vertically, and hence we can transpose a label (i.e. exchange its width and height). Figure 1 shows that folding and transposition can improve the labeling layout: suppose that we represent a character string "GSIS" (acronym of "Graduate School of Information Sciences") in three ways: horizontal, in two lines, and vertical. Each of the Figures 1 (a) – (c) illustrates a label placement using single-shape labels pinned at the left-upper corner. Picture (d) shows by how much the scaling factor increases if three different kinds of shapes can be used at the same time.

Therefore, the following *fixed-corner-position shape-flexible labeling* problem naturally arises:

> Suppose that we have n points and each point has a set of candidate labels of various rectangular shapes pinned at the upper-left corner. The label for each point must be selected from the candidate set and placed after being scaled by σ. What is the largest scaling factor σ that allows us to label all points?

In this paper, we propose a new class of shape-flexible labeling problems, named *Left-part Ordered Flexible Labeling* (LOFL), where the candidate set of rectangles given to each object point must be a *left-part ordered set*. The definition of a left-part ordered set will be given in the next section; a typical example is a set of rectangles pinned at their left-upper corners. Thus, the fixed-corner-position shape-flexible labeling problem is a special case of LOFL.

We show that the decision problem for LOFL can be solved in $O((n+m)\log(n+m))$ time by a simple plane-sweep algorithm. As a consequence, the optimization problem for LOFL can be solved in $O((n+m)\log(n+m)\log\Gamma)$ time if the coordinate values of points are represented by $\log\Gamma$-bit integers. We also give an $O((n+m)\log^2(n+m))$ time algorithm for the optimization problem. To design this algorithm, we use the parametric search paradigm, where we use parallel sort and point location query to design a "guide algorithm" required for the parametric search.

Our method can be used as a subroutine in heuristic algorithms for a practical labeling system. We have done a preliminary experiment on the ability of LOFL to enlarge the labeling size compared to single-shaped models.

This paper is based on an earlier version presented at 8th International Symposium on Graph Drawing.[18]

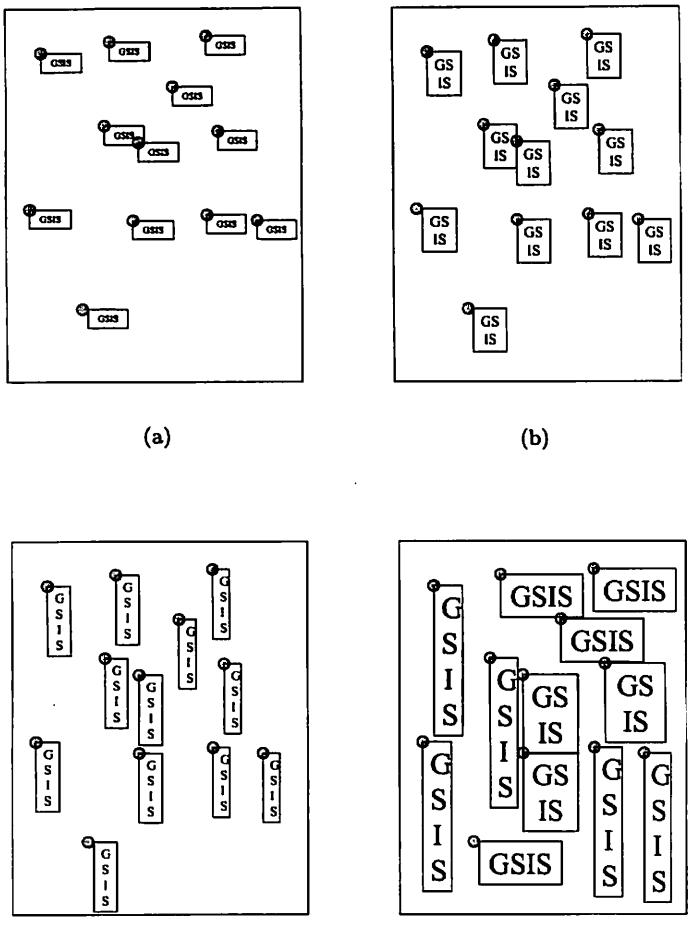

Fig. 1. Placements using three kinds of labels. Each of (a)–(c) uses a single kind of label, whereas (d) uses three kinds of labels.

2. Preliminaries

We assume that rectangles in this paper are open sets; i.e., a rectangle may touch other rectangles and/or obstacles.

A set $\mathcal{R}$ of rectangles in the plane is *totally ordered* with respect to inclusion if any pair R and R' of rectangles in $\mathcal{R}$ satisfies either $R \subseteq R'$ or $R' \subseteq R$.

For a rectangle L representing a label, we fix a point $q(L)$, which we call the *pinning point* of L, in the closure of L. The label is placed on the plane so that its

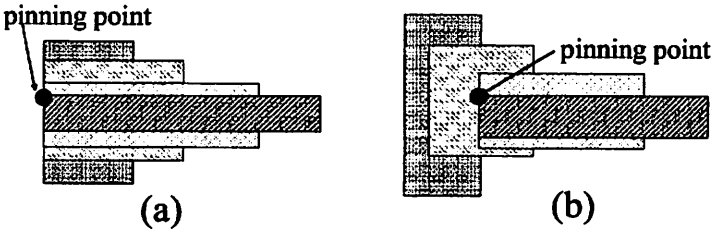

Fig. 2. (a) A degenerate left-part ordered set consisting of four rectangles, and (b) a non-degenerate left-part ordered set.

pinning point is at the corresponding object point.

2.1. *Left-part Ordered Sets*

Consider a set $\mathcal{L}$ of rectangles with pinning points as a set of candidate labels. Suppose that we place the rectangles in $\mathcal{L}$ so that their pinning points are translated to the origin. Let H^- be the closed halfplane defined by $x \leq 0$. $L \cap H^-$ (or its translated/scaled copy if L is translated/scaled) is called the *left part* of L.

Then $\mathcal{L}^- = \{L \cap H^- | L \in \mathcal{L}\}$ is a set of rectangles. If $\mathcal{L}^-$ is totally ordered, we say the set $\mathcal{L}$ satisfies the *left-part ordered property*, and call the set $\mathcal{L}$ a *left-part ordered set.*

We say that a left-part ordered set $\mathcal{L}$ is *degenerate* if the pinning point of each label $L \in \mathcal{L}$ is on the left edge; in other words, $L \cap H^-$ is a vertical segment. Otherwise we say $\mathcal{L}$ is non-degenerate. Figure 2 (a) is an example of a degenerate left-part ordered set, and Figure 2 (b) is an example of a non-degenerate left-part ordered set. For simplicity, we mainly consider degenerate left-part ordered sets to describe algorithms and proofs, since it is routine to generalize them to the non-degenerate case.

We are given a set $P = \{p_1, p_2, \ldots, p_n\}$ of n object points on the plane. Let $p_i = (x_i, y_i)$ for $i = 1, 2, \ldots, n$. We also consider a set Q of polygonal (not necessarily rectilinear) obstacles in the plane, which no label is permitted to overlap. We permit labels to touch obstacles. We assume that the obstacles do not intersect each other. Let m be the total number of edges of polygons in Q. A left-part ordered set $\mathcal{L}_i$ of rectangular labels is given to each object point $p_i \in P$. $\mathcal{L}_i$ and $\mathcal{L}_j$ may be different from each other if $i \neq j$.

We choose a suitable label L_i from $\mathcal{L}_i$ for each $p_i \in P$, scale L_i by the factor σ, and place it in the plane so that its pinning point is placed at p_i. The placement $\mathbf{L} = \{L_1, L_2, \ldots, L_n\}$ is called *feasible* if (1) no two labels overlap each other and (2) no label overlaps any obstacle. Our aim is to find the largest scaling factor σ for which a feasible placement exists, and to compute such a placement.

We assume that there is no pair of labels L and L' in $\mathcal{L}_i$ such that $L \subseteq L'$,

since we need not use the larger label L' (called *redundant label*) in our solution. If we have a label set with such pairs, we preprocess the label sets by removing all redundant labels.

We define an order $<$ on the set $\mathcal{L}_i$ as follows: $L' < L$ for two labels L and L' in $\mathcal{L}_i$ if and only if $L \cap H^- \supset L' \cap H^-$. We say that L' is *left-smaller* than L if $L' < L$.

Using the order, we can naturally give a lexicographical order among the set of feasible solutions (for a fixed σ) as follows: We sort the object points $p_1, p_2, \ldots, p_n$ in non-increasing order with respect to the x-coordinates, and re-arrange the numbering; hence, p_1 is the rightmost point and p_n is the leftmost point. Let $\mathbf{L} = (L_1, L_2, ..., L_n)$ and $\mathbf{L}' = (L'_1, L'_2, ..., L'_n)$ be two different feasible placements where $L_i \in \mathcal{L}_i$ and $L'_i \in \mathcal{L}_i$ are labels for p_i, $1 \leq i \leq n$. Then we define $\mathbf{L}' < \mathbf{L}$ if there is an index j such that $L'_j < L'_j$ and $L'_i = L_i$ for every $i < j$. The minimum feasible placement with respect to this order is called the *left-minimum* solution.

Let $N = \sum_{i=1}^{n} |\mathcal{L}_i|$: N is the number of all candidate rectangles. In Geographic Information System, the cardinality of a left-part ordered set for a point is usually bounded by a constant. Therefore, we assume $N = O(n)$ in this paper, although it is not difficult to generalize our argument to the cases where N is much larger than n, except for the parametric search method in Section 4.3. Indeed, the decision problem can be solved without increasing the time complexity even if the label set of each point is an infinite set, provided that we have an efficient (to be precise, $O(\log(n+m))$ amortized time) method to query the left-smallest label that does not intersect the "frontier" defined in Section 3.2. A typical example is the fixed-corner-position elastic labeling problem,[10,11] where each label set consists of all rectangles with the same area and each label is pinned at its upper-left corner. Another example is the *one-slider model* labeling problem proposed by van Kreveld et al..[14]

3. Decision Problem

3.1. *Algorithm for the Decision Problem*

In this section, we present an $O((n+m)\log(n+m))$ time algorithm to solve the decision problem for a fixed scaling factor σ. Without loss of generality, we may assume $\sigma = 1$ in this section. We first sort the object points in non-increasing order with respect to the x-coordinate values, and re-arrange the numbering as we noted before. We start with the following observation:

Lemma 1. *Let $\mathbf{L} = (L_1, L_2, \ldots, L_n)$ be a feasible placement, and let $1 \leq i \leq n$. If there is a label $L \in \mathcal{L}_i$ which is left-smaller than L_i and intersects none of the labels $L_1, L_2, \ldots, L_{i-1}$ and the obstacles, then the placement $\mathbf{L}' = (L_1, L_2, \ldots, L_{i-1}, L, L_{i+1}, \ldots, L_n)$ that is obtained by replacing L_i by L is also feasible.*

Proof. Suppose for a contradiction that the placement $\mathbf{L}'$ is infeasible. Then there must be an index $j > i$ such that the label L_j assigned to p_j in the original

placement $\mathbf{L}$ intersects L. Since p_j is located to the left of p_i, L_j must intersect the left-part of L. However, L_j must also intersect the left-part of L_i because L is left-smaller than L_i. This contradicts the feasibility of the original placement $\mathbf{L}$. □

From this observation, we can design a simple incremental algorithm, named DECIDE, to decide the feasibility. The input of the algorithm consists of the set of n object points, the set of obstacles, the set $\mathcal{L}_i$ for each $i = 1, 2, \ldots, n$, and a fixed scaling factor σ that is set to be 1 for simplicity. The algorithm outputs the left-minimum solution if the input instance is feasible; otherwise, tells that the input instance is infeasible.

Algorithm *DECIDE*
(*decide the feasibility of a LOFL instance)
1. **for** $i \leftarrow 1$ **to** n
2. **do**
3. **if** every label in $\mathcal{L}_i$ overlaps an obstacle or a label placed so far
4. **then return** "The instance is infeasible"
5. **else** assign p_i the left-smallest label L_i that overlaps neither any obstacle nor any label placed so far
6. **fi**
7. **end**
8. **return** $\mathbf{L} = (L_1, L_2, \ldots, L_n)$ as a feasible solution

3.2. *Implementation and Analysis of the Algorithm*

It is clear that DECIDE is correct. We give an implementation of DECIDE by using a standard plane sweep method, and show that it takes $O((n+m)\log(n+m))$ time. We remark that the plane sweep method is widely used for the rectangle placement and labeling problems; See for example, van Kreveld *et al.*. [14]

First, we assume for simplicity that there is no obstacle, and give an algorithm for that case; we will briefly explain later how to modify it for the case with obstacles. For a label L, its *right-part width* is the horizontal distance between its pinning point and its right edge. As a preprocessing, we sort the rectangles in $\mathcal{L}_i$ in descending order of right-part width for each $i = 1, 2, \ldots, n$. This takes $O(N \log N) = O(n \log n)$ time. For each set $\mathcal{L}_i$ and a positive real number w, let $L(i, w)$ be the left-smallest label in $\mathcal{L}_i$ whose right-part width is at most w.

Let $\mathbf{U}$ be a set of geometric objects (in our case, placed labels) in the plane, and let l be a vertical line, then we say that a point q in an object in $\mathbf{U}$ is *left-visible* from l if q is on the right of l and the horizontal half-line emanating from q to the left does not intersect any objects of $\mathbf{U}$ until it meets l. The union of all left-visible points of objects in $\mathbf{U}$ is called the *frontier* of $\mathbf{U}$ at l.

We denote by $\mathbf{L}[i]$ the labeling of points $p_1, p_2, \ldots, p_i$ obtained by the algorithm DECIDE. Recall that $p_i = (x_i, y_i)$. For t, $x_i > t \geq x_{i+1}$, the frontier of $\mathbf{L}[i]$ at the vertical line $x = t$ is the union of all left-visible segments on left edges of labels

(rectangles) in $\mathbf{L}[i]$. The frontier has $O(i)$ segments, and its orthogonal projection onto the y-axis induces a partition of the y axis into $O(i)$ intervals (to be precise into at most $2i + 1$ intervals). The sorted list $Proj(\mathbf{L}[i])$ of these intervals with respect to the y-coordinate values of the endpoints is called the *projected frontier*. To each interval I in the projected frontier, we assign the x-coordinate value x_I of the segment in the frontier whose projected image is I. The value x_I is set to be ∞ if there is no label in $\mathbf{L}[i]$ whose projection contains I. We implement the list $Proj(\mathbf{L}[i])$ of intervals by using a suitable dynamic binary-search data structure.[17] Thus, we can find the interval I containing y_{i+1} in time $O(\log n)$, and find the next and the previous elements of a currently pointed element in the list in $O(\log n)$ time [a].

Our plane sweep algorithm moves the sweep line $x = t$ to the left from $t = \infty$ to $t = -\infty$. While $x_i > t \geq x_{i+1}$, we maintain the projected frontier $Proj(\mathbf{L}[i])$ together with the values x_I for all intervals I.

Figure 3 illustrates the frontier and the projected frontier. When the sweep line comes to $t = x_{i+1}$, we insert a label of p_{i+1} to $\mathbf{L}[i]$, and update $Proj(\mathbf{L}[i])$ to $Proj(\mathbf{L}[i+1])$.

To be precise, we first search the interval $I \in Proj(\mathbf{L}[i])$ containing the y-coordinate value y_{i+1} of $p_{i+1} = (x_{i+1}, y_{i+1})$, and find the value $x(I)$. We want to find the left-smallest label in $\mathcal{L}_{i+1}$ which does not intersect the frontier. For the purpose, we compute the difference $w = x(I) - x_{i+1}$, and find the rectangle $L = L(i+1, w) \in \mathcal{L}_{i+1}$. Suppose that the right-part width of L is w', and its upper edge and lower edge are located on $y_0(L)$ and $y_1(L)$, respectively. The definition of $L(i+1, w)$ implies $w' \leq w$. We scan the list $Proj(\mathbf{L}[i])$ of intervals in the so-called dovetailing fashion [b] until either we encounter an interval $J \in \mathbf{L}[i]$ such that $x(J) - x_{i+1} < w'$ or we reach both the y-values $y_0(L)$ and $y_1(L)$. In the former case, L intersects the left side of the placed label visible from J, and hence we replace w by $x(J) - x_{i+1}$, find the label $L = L(i+1, w)$ for this new value w, and continue scanning the list for this new label L from the current two positions in the dovetailing scan. In the latter case, we successfully place $L = L(i+1, w)$, and update $Proj(\mathbf{L}[i])$ to $Proj(\mathbf{L}[i+1])$ by inserting the left edge of L; hence, all intervals completely hidden by L are deleted, and at most two intervals partially hidden by the projected image of the left edge of L are replaced by shorter intervals.

Proposition 1. *The time complexity of the algorithm DECIDE is* $O(n \log n)$.

The decision problem is at least as difficult as the element uniqueness problem,[21] and hence the $O(n \log n)$ time complexity is optimal in the algebraic decision tree model.

[a] We can improve the time complexity for finding the next element to $O(1)$ if we use a more sophisticated data structure such as the skip list, although $O(\log n)$ time is enough for our analysis.
[b] In the dovetailing scan (often called *tandem search*), we have two pointers to the list of intervals, and move them from the starting interval I alternatively, one forward and the other backward.

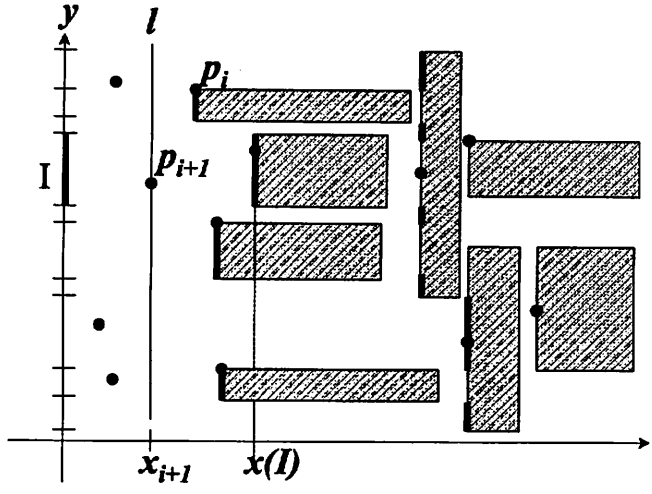

Fig. 3. Frontier of L[i], and the sweep line $l : x = t$ at $t = x_{i+1}$.

If there are obstacles, the framework of the algorithm is the same as the one described above. However, the frontier contains parts of obstacles as well as left edges of placed labels. Each such part may be a segment of an edge of an obstacle or a connected component of the intersection of an obstacle and the sweep line. If the part is a segment of an edge, the corresponding interval in the projected frontier points to the equation of the line that contains the edge.

A major difficulty is in the handling of obstacles intersecting the sweep line. For each connected component of the intersection of obstacles with the sweep line, we have an interval, which changes linearly in the parameter t, in the projected frontier. Such an interval or its adjacent interval may shrink and be eliminated. Therefore, we need to consider a new type of event where an interval in the projected frontier is eliminated. The number of such events is $O(n + m)$. We maintain a priority queue to query the earliest elimination time of intervals in the projected frontier. This priority queue can be updated in $O(\log(m + n))$ time per each of insertion, deletion and replace operations of intervals. Analogously to the analysis in the previous subsection, we can prove that $O(n + m)$ intervals in the projected frontier are inserted, deleted, or replaced in the whole plane sweep procedure. Hence, we can maintain the priority queue in $O((n + m) \log(n + m))$ time in total.

We remark that for a non-degenerate LOFL, a label may intersect an obstacle to the left of the sweep line, and cause a difficulty for the algorithm. However, we can easily resolve this difficulty as follows: Since we place the left-smallest feasible label for a point p_i on the sweep line in the algorithm, if the label intersects an obstacle to the right of the sweep line, any left-larger label intersects this obstacel, and hence

the instance is infeasible. Hence, we run the algorithm without considering the intersection between a label and obstacles to the left of the sweep line, and detect the intersection after we have completely determined the labels for confirming the feasibility.

4. Optimization Problem

4.1. *A Precision-dependent Algorithm*

We consider the problem of finding the maximum feasible value of the scaling factor σ. A simple binary search algorithm on σ works; the algorithm for the decision problem can decide whether we should try a larger σ or a smaller one than the current scaling factor for the next search. If coordinate values of all points are integral and Γ is the maximum of their absolute values, then it suffices to run the algorithm given in Section 3.1 $O(\log \Gamma)$ times (note that $\Gamma^2 \geq n$). Thus, we have:

Theorem 1. *The LOFL optimization problem can be solved in* $O((n+m)\log(n+m)\log\Gamma)$ *time.*

4.2. *Precision-independent Algorithms*

The binary search algorithm above is efficient for practical inputs for which $\log\Gamma = O(\log(n+m))$ holds. However, an efficient algorithm with time complexity independent of Γ is desirable from a theoretical point of view. We design such an $O((n+m)\log^2(n+m))$ time algorithm for the optimization problem.

One possible method is to consider the *conflict graph* of candidate labels, and first list up all possible critical values of σ, and then do a binary search on the list. For several labeling problems,[8] a list of $O(n)$ critical values can be found, and the above method is efficient. This is because the size of the conflict graph at $\sigma = \sigma_{\mathrm{opt}}$ is reduced to $O(n)$ for those problems if "clearly useless" labels can be removed from the instance. Unfortunately, the property does not hold for LOFL, and we do not know how to obtain such a list of size $o(n^2)$. Thus, we pursue another approach.

Meggido's parametric search[16] is a famous method to transform a precision-dependent binary search algorithm into a precision-independent algorithm. The method is especially useful in computational geometry.[20,4]

We give a brief introduction to the parametric search paradigm (see Ref. [20] for details). Suppose that F is a 0-1 valued *monotone* function on a parameter θ: there is a value θ_{opt} such that $F(\theta) = 1$ if $\theta \leq \theta_{\mathrm{opt}}$ and $F(\theta) = 0$ if $\theta > \theta_{\mathrm{opt}}$. Our aim is to compute the value θ_{opt}. Parametric search assumes that the following two algorithms, A and D, for computing $F(\theta)$ for a given value θ are available: The algorithm D is called *decision algorithm*, which is the fastest available algorithm to compute $F(\theta)$. Assume that D takes $O(T_D)$ time. The other algorithm A is called *guide algorithm*, whose computation can be processed the linear decision tree model.

We simulate the behavior of A for $\theta = \theta_{\mathrm{opt}}$ without knowing the value θ_{opt} in cooperation with the decision algorithm D, and find the value θ_{opt} in the course

of the simulation. Indeed, at each decision of A in the linear decision tree, we need to compare θ_{opt} with a threshold value for the decision; then, we run the decision algorithm for substituting the threshold value to θ, and see whether $F(\theta) = 0$ or 1 to know the comparison of the threshold value to θ_{opt}. This process shrinks the parameter interval defined by threshold values containing θ_{opt}, and we can pinpoint θ_{opt} when we complete the simulation.

It is advantageous to use a guide algorithm that has a parallel structure, although we do not use a parallel machine in our computation. If A takes $O(t_A)$ parallel time with M processors, then we can simulate A for $\theta = \theta_{\text{opt}}$ without knowledge of θ_{opt} in $O(t_A M + t_A T_D \log M)$ time. Cole's acceleration method[6] can often improve the time complexity to $O(t_A M + t_A T_D)$.

Let us consider our LOFL problem. We define a monotone function F as follows: $F(\sigma) = 1$ if and only if there is a feasible placement for the scaling factor σ. To do parametric search for the parameter σ, we can use DECIDE as decision algorithm.

Unfortunately, for our problem, a guide algorithm with $t_A = O(\log(n+m))$ and $M = O(n+m)$ seems to be difficult to design. To overcome this difficulty, we adopt a "heterogeneous" version of parametric search. The heterogeneous parametric search paradigm uses a "weaker" guide algorithm A that cannot compute $F(\sigma)$ itself even if σ is given as an input. Instead, A computes another function $G(\sigma)$, where the range of $G(\sigma)$ is not $\{0,1\}$ but a much larger category. The required condition is that $G(\sigma) = G(\sigma')$ always implies $F(\sigma) = F(\sigma')$ for any σ and σ'. Intuitively, G gives a refinement of F. In particular, we will use a guide algorithm consisting of parallel sort and point location algorithms.

The idea of heterogeneous parametric search was implicitly given in Megiddo's paper,[16] in which he solved a problem on the parametric minimum spanning tree of a graph by using a parallel sorting algorithm as its guide algorithm. Cole[6] dealt with the heterogeneous parametric search in which the guide algorithm is a parallel sort using a sorting network.

4.3. *A Parametric Search Algorithm for LOFL*

The preprocessing step for our parametric search algorithm consists of preparing a point location data structure from the set Q of polygonal obstacles as follows: We first construct a triangulation $\mathcal{D}(Q)$ of the plane into $O(m)$ triangles so that each triangle is either contained in an obstacle or lies completely outside all obstacles. All vertices, edges, and triangles in $\mathcal{D}(Q)$ are called *faces* of $\mathcal{D}(Q)$. Then we prepare a point location data structure so that we can find the face of $\mathcal{D}(Q)$ containing a query point in $O(\log m)$ time. The triangulation and the point location data structure can be constructed in $O(m \log m)$ time (e.g. using Ref. [21]), and we do not need to construct it in parallel since it is independent of the value of the scaling factor.

Let $\mathcal{L} = \cup_{i=1}^{n} \mathcal{L}_i$ be the set of all label candidates, and let $S(\sigma)$ be the set of corner points of all rectangles in $\mathcal{L}$ after being scaled by σ and placed so that their

pinning points coincide with the corresponding object points in P. Let $V(Q)$ be the set of all vertices of polygonal obstacles in Q.

Our guide algorithm first sorts two lists $X(S(\sigma) \cup V(Q))$ and $Y(S(\sigma) \cup V(Q))$ of the x- and y-coordinates, respectively, of the point set $S(\sigma) \cup V(Q)$, and then locates all points of $S(\sigma)$ in $\mathcal{D}(Q)$ in parallel.

We call a pair τ and σ of parameter values *equivalent* if (1) $X(S(\sigma) \cup V(Q)) = X(S(\tau) \cup V(Q))$, (2) $Y(S(\sigma) \cup V(Q)) = Y(S(\tau) \cup V(Q))$ and (3) each point in $S(\tau)$ is contained in the same face of $\mathcal{D}(Q)$ as the corresponding point in $S(\sigma)$ is.

Lemma 2. *Let σ and τ be equivalent, then there is a feasible solution of LOFL for the scaling factor τ if and only if there is a feasible solution for σ. Moreover, there is no value $\tau \neq \sigma_{\mathrm{opt}}$ such that τ is equivalent to σ_{opt}.*

Proof. Suppose that there is a feasible solution $\mathcal{L}$ for σ, and τ is equivalent to σ. We claim that $\mathcal{L}$ is also feasible for the scaling factor τ. Assume that $\mathcal{L}$ is not feasible for τ. First, suppose that for a pair of labels L and L' in $\mathcal{L}$, τL intersects $\tau L'$, where τL is the label obtained by scaling L by τ and placed on the map. Since $\mathcal{L}$ is feasible for σ, the labels σL and $\sigma L'$ do not intersect. Hence, either their projected images to the x-axis do not overlap or their projected images to the y-axis do not overlap. Without loss of generality, we assume that the former holds. Since τ and σ are equivalent to each other, the projected images of τL and $\tau L'$ to the x-axis do not overlap each other, and hence $\tau L \cap \tau L' = \emptyset$; a contradiction. Next, suppose that a label τL overlaps with an obstacle. Then, one of the following cases occur: (a) a corner of τL lies in the interior of an obstacle, (b) a vertex of an obstacle is in σL, and (c) τL intersects with a triangle Δ in $\mathcal{D}(Q)$ contained in an obstacle so that the intersection is a trapezoid. For the case (a), σL also satisfies (a) since the corner point of σL lies in the same triangle of $\mathcal{D}(Q)$ as that of τL from the condition (1). For the case (b), σL also satisfies (b) since the x and y coordinate values of the vertex must be in the interval of projection of σL to the x and y axis, respectively, from the conditions (1) and (2), For the case (c), we can assume that the paralell edges of the trapezoid are vertical, and we can derive that σL satisfies (c) from the conditions (1) and (3). Thus, we have a contradiction, too. □

Hence, by simulating our guide algorithm, we can compute σ_{opt}. Sorting of $O(n+m)$ elements can be done by applying an AKS sorting network[2] in $O(\log(n+m))$ parallel time using $O(n+m)$ processors. The point location query can be done in parallel for all of the $O(n)$ corner points of the labele candidates in $O(\log m)$ time. Thus, the guide algorithm runs in $O(\log(n+m))$ steps using $O(n+m)$ processors. Moreover, we can apply Cole's acceleration method.[6] Hence, our parametric search algorithm runs in $O((n+m)\log^2(n+m))$ sequential time. Thus, we have obtained the following theorem:

Theorem 2. *In $O((n+m)\log^2(n+m))$ time, we can find the maximum scaling factor that permits a feasible LOFL of n points in the plane given polygonal obstacles*

with m edges in total.

5. Heuristics Based on LOFL

In a practical GIS system, a map labeling problem is often given in a form that is theoretically NP-hard. Therefore, heuristics methods or hybrid methods are effective in practice. [23,24,25] LOFL can be used as a powerful weapon to design heuristics combined with other methods. Suppose we have a feasible labeling with a scaling factor σ given by some method, and want to improve the factor by changing the shape of labels. Let L_i be the label for $p_i \in P$ in the labeling. In place of the single label L_i, to each $p_i \in P$, we assign an appropriate left-part ordered set $\mathcal{L}_i$ such that $\mathcal{L}_i \ni L_i$. Thus, we have an instance of LOFL. The scaling factor in the solution of this LOFL instance is larger than or equal to σ, and is often much larger than σ. This can be considered as a "local improvement routine," which is an important tool in meta-heuristics.

5.1. *A Heuristic for Two-position LOFL*

Suppose that we are given an instance for which the set of candidate labels for each $p_i \in P$ is the union of two left-part ordered sets $\mathcal{L}_i$ and $\mathcal{M}_i$. We call this model *two-position LOFL*, since it can be regarded as a combination of LOFL and the two-position model.[8,14] The problem is NP-hard to approximate the optimal scaling factor within a ratio of $\frac{1}{2} + \epsilon$; therefore, we need a heuristic. Our heuristic for two-position LOFL is a combination of solutions of LOFL and 2LABEL, where 2LABEL is an algorithm for solving the two-position labeling problem where we have (at most) two candidate rectangles for each object point. Forman and Wagner[8] gave an efficient implementation of 2LABEL.

If we have an oracle to determine from which of $\mathcal{L}_i$ or $\mathcal{M}_i$ the label for p_i, $i = 1, 2, \ldots, n$, should be selected, we can reduce the problem to LOFL. In our heuristic, we use 2LABEL to substitute for such an oracle.

We first select two candidate labels $L_i \in \mathcal{L}_i$ and $M_i \in \mathcal{M}_i$ for each point $p_i \in P$. These two labels can be randomly chosen from the set of candidate labels, or just use the most standard (say, square) labels. Thus, we have an instance of 2LABEL, and we compute its optimal solution. If L_i (resp. M_i) is in the output placement of the 2LABEL instance, we assign the family $\mathcal{L}_i$ (resp. $\mathcal{M}_i$) to p_i. Thus, we have an instance of LOFL.

We can also use LOFL to construct an instance of 2LABEL as follows: Suppose that we have an optimal solution of LOFL for an instance where either $\mathcal{L}_i$ or $\mathcal{M}_i$ is assigned as the set of candidate labels for each $p_i \in P$. If $L_i \in \mathcal{L}_i$ (resp. $M_i \in \mathcal{M}_i$) is the label chosen for $p_i \in P$ in the solution, then we choose a *partner* label M_i (resp. L_i) from $\mathcal{M}_i$ (resp. $\mathcal{L}_i$) and assign a pair (L_i, M_i) of candidate labels for p_i to have an instance of 2LABEL. We choose as partner label a label that intersects the least number of labels in the current solution of LOFL.

Thus, we alternately apply 2LABEL and LOFL until the increase of the scaling factor stops. We can similarly combine the one-slider (vertical slider) model[14] and LOFL.

6. Experimental Results

We have done an experiment to see the ability of LOFL to enlarge the scaling factor. We compared four different labeling models: (1) fixed-position model, (2) two-position model, (3) LOFL, and (4) two-position LOFL (2-LOFL).

To be precise, for each object point, we assign the following candidate labels for the respective labeling models: (1) A left-upper pinned rectangle of height 3σ and width 4σ. (2) A pair of rectangles with height 3σ and breadth 4σ, one of which is left-upper pinned and the other is left-lower pinned[c]. (3) A set of six kinds of left-upper pinned rectangles of area $12\sigma^2$ whose height-breadth ratios are 12, 3, 4/3, 3/4, 1/3, and 1/12, which correspond to factorizations of 12. (4) A set of rectangles consisting of those in (2) and their reflected copies pinned at the left-lower corner.

We apply LOFL instead of Forman-Wagner's algorithm for solving (2). For solving the LOFL ((2) and (3)), we implement the binary-search algorithm given in Section 4.1 instead of the parametric-search algorithm, and only consider integral scaling factors. We applied the same paradigm for solving (1): Do binary search on the scale factor, and apply plane sweep for intersection detection. We programmed in C_{++} using LEDA program library. We adopted a red-black tree for maintaining the frontier.

We randomly generate n integral object points in a square region of size 50000×50000 for each of $n = 20, 40, 60, 80, 100, 200, 400, 800$ and 1600.

Table 1 shows the average scaling factor over 100 instances for each of (1), (2), (3), and (4) for each n. They show that LOFL and 2-position have almost same performance for a small n, whereas 2-position can place larger labels for a large n. 2-LOFL works well for every range of n, and improvement of the label size over 2-position is about 50% – 60%. This means that hybrid of the two methods is effective, although the improvement is not drastic.

Note that the table indicates only the sizes of the labels, and does not indicate the visual quality of labeling outputs, since a labeling with various shapes is often less beautiful than a labeling with a single shape. Indeed, in a practical map labeling instance, only a portion of the point set should be given labels with various shapes.

Table 2 shows the computation time (the total computation time for 100 instances) for each method. The curves representing computation time are sublinear in n, which is not surprizing since average size of the list representing a frontier is smaller than n. Use of a theoretical priority queue is essential, since if we use a simple list instead of a red-black tree, the computation time follows nearly quadratic curve of n, and at $n = 1600$, it takes 299.92 and 2093.68 seconds for LOFL and

[c]An instance is also a LOFL if we rotate it by 90 degrees.

n	fixed	2-position	LOFL	2-LOFL
20	434.00	1041.82	1131.52	1778.46
40	217.35	611.37	637.60	1030.98
60	150.52	460.76	411.14	749.26
80	130.69	375.12	380.07	603.79
100	88.79	279.53	272.82	473.10
200	47.18	176.67	151.35	296.90
400	22.19	104.98	73.37	174.46
800	11.39	55.57	39.16	93.36
1600	5.14	33.95	18.21	48.58

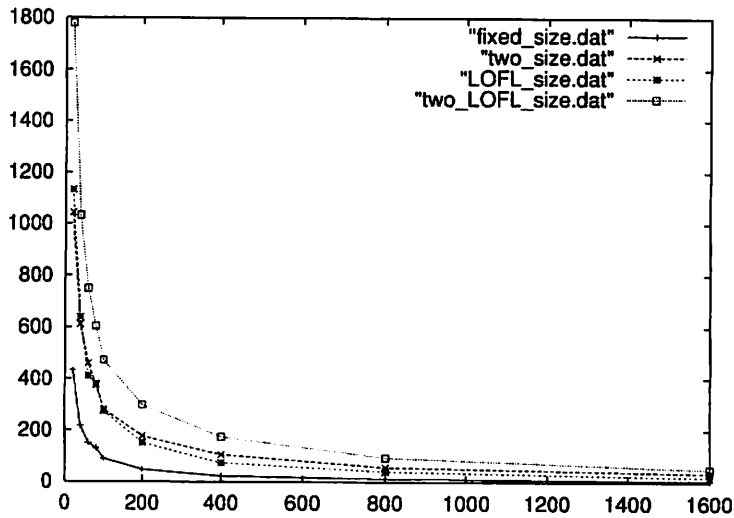

Table 1. Average scaling factor for each method.

2-LOFL, respectively.

We can observe in Table 1 that the average scaling factor for the fixed-position model is approximately $9000n^{-1}$, which can be predicted mathematicaly as follows: We insert points randomly in a unit square and give them labels of area $A = 12\sigma^2/50000^2$ accoding to the fixed position model until we find overlapping between labels. Suppose that we have placed i rectangular labels and insert a new point p. For each existing point q, if p is in the extended rectangle of area $4A$ around q, we have overlapping between labels. The area of the union of the extended rectangles is at most $4iA$ and its expectation is at least $4iA(1 - 4iA)$; thus, we approximate it by $4iA$ since $4iA \ll 1$. Therefore, the area of the region we can place p is $1 - 4iA$, and the probability that random n points can be labeled with the scaling factor σ is $F(\sigma) = \prod_{i=1}^{n-1}(1 - 4iA) \sim e^{-2An(n-1)} = e^{-24\sigma^2 n(n-1)/50000^2}$. Thus, the expected value E of the maximum scaling factor σ is $\int_0^\infty -\sigma \frac{dF(\sigma)}{d\sigma} d\sigma$. By using

n	fixed	2-position	LOFL	2-LOFL
20	0.10	0.08	0.08	1.77
40	0.22	0.26	0.21	2.76
60	0.18	0.28	0.35	3.89
80	0.21	0.38	0.47	4.98
100	0.44	0.47	0.44	5.95
200	0.78	0.92	0.87	10.97
400	1.28	1.73	1.69	20.70
800	2.35	3.17	2.83	38.37
1600	3.94	5.82	5.22	70.22

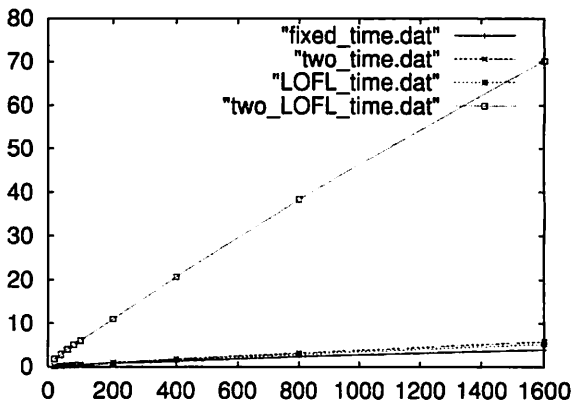

Table 2. Computation time (in seconds).

a well known formula $\int_0^\infty e^{-b^2x^2}dx = \sqrt{\pi}/2|b|$, it is routine calculation to have $E = 50000\sqrt{\pi/96n(n-1)} \sim 9000n^{-1}$.

For other three methods, the ratio of the scaling factor to n^{-1} becomes larger when n grows.

7. Concluding Remarks

LOFL is one of few polynomial-time soluble labeling schemes. It is simple, flexible, and efficient. In practical map labeling solution, we need to design heuristics algorithms in order to place all the labels efficiently.[5,23] For the purpose, we need to have various efficient algorithms as building blocks. We believe that LOFL is a strong algorithmic tool, and hope it will be utilized in practical geographic information systems. It is open problem to give generalization of LOFL and its three-dimensional variation.

Acknowledgements

The authors thank anonymous referees for several kind comments and suggestions on the initial version of this paper. Especially, they informed a lot of references on map labeling problems.

References

1. P. Agarwal, M. van Kreveld, and S. Suri, Label placement by maximum independent set in rectangles, *Computational Geometry, Theory and Applications*, **11** (1998), pp. 209–218.
2. M. Ajtai, J. Komlos, and E. Szemeredi, Sorting in $c\log(n)$ parallel steps, *Combinatorica*, **3** (1983), pp.1–19.
3. H. Aonuma, H. Imai, K. Imai, and T. Tokuyama, Maximin locations of convex objects in a polygon and related dynamic Voronoi diagrams, *Proc. 6th Annual ACM Symp. on Computational Geometry* (1990), pp. 225–234.
4. P. Agarwal and M. Sharir, Efficient algorithms for geometric optimization, *ACM Comput. Surv.*, **30** (1998), pp. 412-458.
5. J. Christensen, S. Friedman, J. Marks, and S. Shieber, Empirical testing of algorithms for variable-sized label placement. *Proc. 13th Annual ACM Symp. on Computational Geometry* (1997), pp. 415–417.
6. R. Cole, Slowing down sorting network to obtain faster sorting algorithms, *J. ACM*, **34** (1987), pp.200–208.
7. R. Duncan, J. Quian, and B. Zhu, Polynomial time algorithms for three-label point labeling, *Proc. 7th Annual Int. Computing and Combinatorial Conf. (COCOON'01), vol. 2108 of LNCS* (2000), pp. 191-200.
8. M. Formann and F. Wagner, A packing problem with applications to lettering of maps, *Proc. 7th Annual ACM Symp. on Computational Geometry* (1991), pp. 281–290.
9. M. Garrido, C. Iturriaga, A. Márquez, J. Portillo, P. Reyes, and A. Wolf, Lableing subway lines, *Proc. 12th Annual Int. Symp. on Algorithms and Computation. (ISAAC'01), vol. 2223 of LNCS* (2001), pp. 649–659.
10. C. Iturriaga, Map Labeling Problems, *Ph.D Thesis*, Waterloo University, 1999.
11. C. Iturriaga and A. Lubiw, Elastic labels around the perimeter of a map, *Proc. WADS'99* (1999), pp. 306–317.
12. K. Kakoulis and I. Tollis, A unified approach to labeling graphical features, *Proc. 14th Annual ACM Symp. on Computational Geometry* (1998), pp. 347–356.
13. K. Kato, Studies on the Geometric Location Problems, L1 Approximation and Character Placing, Master Thesis, Kyushu University (February 1989).
14. M. van Kreveld, T. Strijk, and A. Wolff, Point set labeling with sliding labels, *Computational Geometry, Theory and Applications*, **13** (1999), pp. 21–47.
15. S.-K. Kim, C-S. Shin, and T.-C. Yang, Labeling a rectilinear map with sliding labels, *International Journal of Computational Geometry and Applications* **11** (2001), pp.167–179.
16. N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, *J. ACM*, **30** (1983), pp. 852–865.
17. K. Mehlhorn, *Data Structures and Algorithms 1: Sorting and Searching*, ETACS Monograph 1, Springer Verlag, 1984.
18. S. Nakano, T. Nishizeki, T. Tokuyama, and S. Watanabe, Labeling points with rectangles of various shapes, *Graph Drawing, Proc. 8th Int. Symp. (GD2000), vol 1984 of LNCS* (2001), pp. 91–102.

19. Z. Qin, A. Wolf, Y. Xu. and B. Zhu, New algoirthms for tow-label point labeling, *Proc. 8th Europ. Symp. on Algorithms (ESA'00), vol. 1879 of LNCS* (2000) pp. 368–379.
20. J. Salowe, Parametric search, Section 37 of *Handbook of Discrete and Computational Geometry*, pp. 683–695, (ed. J. Goodman and R. Polack), (1997) CRC Press.
21. I. Shamos and F. Preparata, *Computational Geometry – An Introduction*, Springer Verlag, 1985.
22. T. Strijk and M. van Kreveld, Labeling a rectilinear map more efficiently, *Inform. Processing Letters* **69** (1999), pp. 25–30.
23. F. Wagner and A. Wolff, A practical map labeling heuristics algorithm *Computational Geometry, Theory and Applications*, **7** (1997), pp. 387–404.
24. F. Wagner and A. Wolff, A combinatorial framework for map labeling, *Graph Drawing, Proc. 6th Int. Symp. (GD'98) vol. 1547 of LNCS* (1998), pp. 316–331.
25. M. Yamamoto, G. Camara and L. Lorena, Tabu search heuristics for point-feature cartographical label placement, *GeoInformatica* (2000) (also see http://www.lac.inpe.br/~lorena/missae/index.html)