Labeling Points with Rectangles of Various Shapes

Shin-ichi Nakano[1], Takao Nishizeki[2], Takeshi Tokuyama[2], and Shuhei Watanabe[2]

[1] Department of Information Engineering, Gunma University, Kiryu 376-8515, Japan, Email: nakano@cs.gunma-u.ac.jp
[2] Graduate School of Information Sciences, Tohoku University, Aobayama, Sendai 980-8579, Japan. Emails: (nabe,nishi)@nishizeki.ecei.tohoku.ac.jp, tokuyama@dais.is.tohoku.ac.jp

Abstract. We deal with a map-labeling problem, named LOFL (Left-part Ordered Flexible Labeling), to label a set of points in a plane with 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 number of edges of polygonal obstacles.

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 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 a database of sets of points representing locations of objects together with labels of the objects, and should have a function to insert labels to a non-labeled map efficiently.

The problem of locating labels in a map is called the map labeling (or map lettering) [9,14,15,16]. Approximating a label (a string of characters) by its bounding rectangle, one can formulate the map-labeling problem as the problem of locating 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 to the object, (2) rectangles do not overlap each other, and (3) each rectangle does not overlap any obstacle in the map. The condition (1) will be mathematically formulated 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

J. Marks (Ed.): GD 2000, LNCS 1984, pp. 91–102, 2001.

is often chosen as an object point. Moreover, we only consider axis parallel rectangles as labels. See [3,7] for more complicated 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 the conditions (1), (2), and (3) above. Such a problem is called the *decision problem*. We also want to consider the *optimization problem* in which we compute the maximum character size σ, called a *scaling factor*, for which there is a feasible solution. For convenience's sake, we assume that rectangles are closed, but we allow a rectangle to touch other rectangles or obstacles on its boundary.

The decision problem is hard in general: Formann and Wagner [5] showed that if there are four candidates of the placement for each label, 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 [8] showed that the problem remains to be NP-hard if there are three candidates for each label.

On the other side, 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 [5]. Moreover, approximation algorithms with provable approximation ratios are given for several useful versions of the map labeling [9,14]. 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 [1,9].

We deal with another type of a map labeling, called a *shape-flexible labeling*, where we can flexibly choose the shape of each label from a candidate set of rectangles. The chosen labels are placed in the map after scaled by a scaling factor σ which is common 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 candidate sets. If each candidate set is the set of all rectangles with a given area, the labeling is called an *elastic labeling*, and some special cases were investigated by Iturriaga and Lubiw [6].

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). It is often seen that folding and transposition can improve the labeling layout: Suppose that we represent a character string "GSIS"[1] by using three ways: horizontal, in two lines, and vertical. Each of the first three pictures (a), (b), and (c) of Figure 1 illustrates a label placement using single-shape labels

[1] acronym of "Graduate School of Information Sciences"

pinned at the left-upper corner. The picture (d) shows the improvement of the scaling factor if we can use three different kinds of shapes all together.

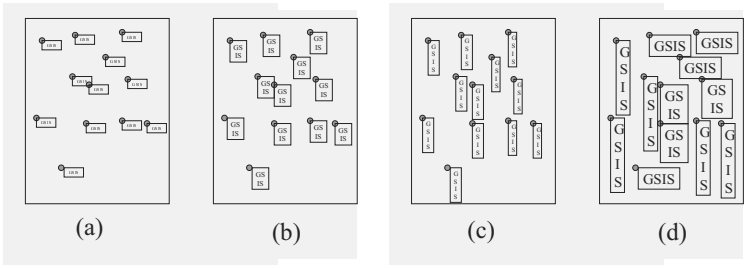

Fig. 1. Placements using three kinds of labels. Each of (a),(b), and (c) uses a single kind of labels, whereas (d) uses three kinds of labels.

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 scaled by σ. How difficult is it to find the largest scaling factor σ 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 in a novel way; 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.

2 Preliminaries

2.1 Left-Part Ordered Sets

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. 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 word, $L \cap H^-$ 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 non-degenerate left-part ordered set. For simplicity, we mainly consider a degenerate left-part ordered set to describe algorithms and proofs, since it is routine to generalize them for a non-degenerate case.

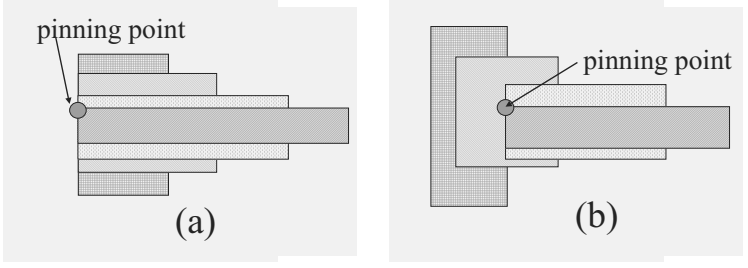

Fig. 2. (a) A degenerate left-part ordered set consisting of four rectangles, and (b) a nondegenerate left-part ordered set.

We are given a set $P = \{p_1, p_2, \ldots, p_n\}$ of n object points on the plane. Let $p_i = (x_i, y_j)$ 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 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 $i \neq j$.

Let σ be a positive real value, called a *scaling factor*. 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 located 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 the placement.

We assume that there is no pair of labels L and L' in $\mathcal{L}_i$ such that $L \subset L'$, since we need not use the larger label L' (called a *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 nonincreasing order with respect to the x-coordinates, and rearrange 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. 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 (in 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, where each label set consists of all rectangles with same area and each label is pinned at its upper-left corner. Another example is the *1-slider model* labeling problem proposed by van Kreveld et al.[9].

2.2 Approximation Hardness for Label Sets without Left-Part Ordered Property

Before presenting algorithms for LOFL, we remark that the left-part ordered property is crucial for designing a polynomial time algorithm. Indeed, it is NP-hard to compute a feasible placement whose scaling factor is larger than $\frac{1}{2} + \epsilon$ times the optimal scaling factor for any positive constant ϵ. The hardness result can be easily obtained by modifying the reduction of the planar 3SAT problem to a labeling problem with three candidate labels [8], and hence the proof is omitted in this version. On the other hand, one cannot construct an "alternating cycle" gadget representing a graph-edge in the reduction if only a left-part ordered set is allowed to each point.

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 σ. Without loss of generality, we may assume $\sigma = 1$ in this section. We first sort the object points in nonincreasing 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 we can replace L_i by L to obtain another feasible placement $\mathbf{L}'$, where $\mathbf{L}' = (L_1, L_2, \ldots, L_{i-1}, L, L_{i+1}, \ldots, L_n)$.*

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. It is clear that DECIDE is correct, and it outputs the left-minimum solution if the input instance is feasible:

Algorithm *DECIDE*
(*decide the feasibility of a LOFL instance for a given σ)
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 problem 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

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 in the rectangle placement and labeling problems; See for example, van Kreveld *et al.* [9].

First, we assume for simplicity that there is no obstacle, and give an algorithm for the 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 $i = 1, 2, \ldots, n$. It 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 a 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 (in precise, 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 [11]. Thus, we can find the interval I containing y_{i+1} in time $O(\log n)$, and scan the list of intervals in $O(\log n)$ time per interval.

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. 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 $Proj(\mathbf{L}[i+1])$. We omit details in this version because of space limitation.

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

The decision problem is at least as difficult as the element uniqueness problem [13], and hence the $O(n \log n)$ time complexity is optimal on the algebraic decision tree model.

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

A major difficulty is in 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 linearly changes 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 events where an interval in the projected frontier is eliminated. However, 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 cut-down 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 cut down in the whole plane sweep procedure. Hence, we can maintain the priority queue in $O((n+m)\log(n+m))$ time in total.

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 value, then it suffices to run the algorithm given in Section 3.1 an $O(\log \Gamma)$ number of times (note that $\Gamma^2 \geq n$). Thus, we have:

Theorem 1. *The optimization problem of LOFL 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 the 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 apply the binary search to the list. For several labeling problems [5], a list of size $O(n)$ of the critical values can be found, and the above method is efficient. This is because the size of the conflict graph at $\sigma = \sigma_{opt}$ is reduced to $O(n)$ for those problems if "clearly useless" labels are omitted. Unfortunately, the property does not hold for the LOFL, and we do not know how to obtain such a list of size $o(n^2)$. Thus, we apply another approach.

Meggido's parametric search [10] is a famous method to transform a precision-dependent binary search algorithm into a precision-independent algorithm. Especially, the method is quite useful in computational geometry [12].

We give a brief introduction to the parametric search paradigm (see [12] 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 $\theta \leq \theta_{opt}$ and $F(\theta) = 0$ if $\theta > \theta_{opt}$. 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 a *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 a *guide algorithm*. We simulate the behavior of A for $\theta = \theta_{opt}$ 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. 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 $\theta = \theta_{opt}$ without inputting the value θ_{opt} in $O(t_A M \log M + t_A T_D \log M)$ sequential time. Cole's acceleration method [4] can often improve the time complexity to $O(t_A M \log M + t_A T_D)$.

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. We can use the parametric search paradigm regarding σ as the parameter. We use DECIDE for the 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 the 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 is 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 query algorithms.

The idea of the heterogeneous parametric search was implicitly given in Megiddo's paper [10], 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 [4] dealt with the heterogeneous parametric search in which the guide algorithm is a parallel sort using a sorting network. However, to the author's knowledge, this is the first time that a heterogeneous parametric search algorithm using a guide algorithm involving a computational geometric procedure is proposed.

4.3 Parametric Search Algorithm for LOFL

As preprocessing of our parametric search algorithm, we prepare 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 completely outside 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. [13]), 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 candidate labels, and let $S(\sigma)$ be the set of corner points of rectangles in $\mathcal{L}$ after scaled by σ and placed so that the pinning points come to their corresponding object points in P. Let $V(Q)$ be the set of all vertices of polygonal obstacles in Q.

Our guide algorithm first computes the sorting lists $X(S(\sigma) \cup V(Q))$ and $Y(S(\sigma) \cup V(Q))$ of the point set $S(\sigma) \cup V(Q)$ with respect to x- and y-coordinate values, and then locates all points of $S(\sigma)$ in $\mathcal{D}(Q)$ in parallel.

A pair τ and σ of parameter values are called *equivalent* to each other 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 $\tau \neq \sigma_{opt}$ such that τ is equivalent to σ_{opt}.*

Hence, by simulating our guide algorithm, we can compute σ_{opt}. Sorting of $O(n + m)$ elements can be done by applying 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 each of $O(n)$ points in $O(\log m)$ time. Thus, the guide algorithm runs in $O(\log(n + m))$ time using $O(n + m)$ processors, Moreover, we can apply Cole's acceleration method [4]. 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 permitting a feasible labeling of LOFL of n points in a plane with polygonal obstacles of m edges.*

5 Heuristics by Using 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 often effective in practice [14,15,16]. 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 Algorithm for the Two-Position LOFL

Suppose that we are given an instance for which the set of candidate labels for each $p_i \in P$ is a 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 [5,9]. As we have noted before, the problem is NP-hard to approximate the optimal scaling factor within a ratio $\frac{1}{2} + \epsilon$; therefore,

we need a heuristic. We solve the two-position LOFL by using 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 [5] 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 can also use LOFL to construct an instance of 2LABEL. We omit details in this version. 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 [9] and LOFL.

6 Preliminary Experimental Results

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

In precise, for each object point, we assign the following candidate labels for the respective labeling models: (1) A left-upper pinned rectangle with height 3σ and breadth 4σ. (2) 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 (they correspond to factorizations of 12 to 12×1, 6×2 and 4×3). (3) A pair of rectangles with height 3σ and breadth 4σ, one of which is left-upper pinned and the other is left-lower pinned[2]. (4) A set of rectangles consisting of those in (2) and their reflected copies pinned at the left-lower corner.

We randomly generate n integral object points in a square region of size 50000×50000 for each of $n = 20, 40, 60$, and 80. We did not place obstacles. Table 1 shows the average scaling factor over 1000 instances for each of (1), (2), (3), and (4) for each of n. Note that the table does not indicate the quality of labeling outputs: Although (2) can use labels with a larger area than (3), it does not say that it is better than (3), 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.

7 Concluding Remarks

Our current implementation of LOFL in the experiment is rather naive; We are preparing for an experiment by using larger and practical instances, and performance of algorithms will be reported there.

In practical applications, we often want to have an algorithm to place as many labels as possible for a given instance of LOFL which is infeasible for a

[2] It is also a LOFL if we rotate the instance by 90°.

Table 1. Scaling factors of the labeling models

	fixed-position	LOFL	two-position	two-position LOFL
n=20	447	1146	1044	1477
n=40	235	643	550	835
n=60	155	445	359	598
n=80	115	333	250	467

fixed scaling factor σ. Design of efficient algorithms or heuristics for this problem is an important future problem.

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) 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 ACM Symp. on Computational Geometry* (1990) 225–234.
4. R. Cole, Slowing down sorting network to obtain faster sorting algorithms, *J. ACM,* **34** (1987) 200–208.
5. M. Formann and F. Wagner, A packing problem with applications to lettering of maps, *Proc. 7th ACM Symp. on Computational Geometry* (1991) 281–290.
6. C. Iturriaga and A. Lubiw, Elastic labels around the perimeter of a map, *Proc. WADS'99* (1999) 306–317
7. K. Kakoulis and I. Tollis, A unified approach to labeling graphical features, *Proc. 14th ACM Symp. on Computational Geometry* (1998) 347–356.
8. K. Kato, Studies on the Geometric Location Problems, L1 Approximation and Character Placing, Master Thesis, Kyushu University (February 1989).
9. M. van Kreveld, T. Strijk, and A. Wolff, Point set labeling with sliding labels, *Computational Geometry, Theory and Applications,* **13** (1999) 21–47.
10. N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, *J. ACM,* **30** (1983) 852–865.
11. K. Mehlhorn, *Data Structures and Algorithms 1: Sorting and Searching,* ETACS Monograph 1, Springer Verlag, 1984.
12. J. Salowe, Parametric search, Section 37 of *Handbook of Discrete and Computational Geometry*, 683–695, (ed. J. Goodman and R. Polack), (1997) CRC Press.
13. I. Shamos and F. Preparata, *Computational Geometry – An Introduction*, Springer Verlag, 1985.
14. F. Wagner and A. Wolff, A practical map labeling heuristics algorithm *Computational Geometry, Theory and Applications,* **7** (1997) 387–404.
15. F. Wagner and A. Wolff, A combinatorial framework for map labeling, *Proc. Graph Drawing '98,* LNCS 1547 (1998) 316–331.
16. M. Yamamoto, G. Camara, L. Lorena, Tabu search heuristics for point-feature cartographical label placement, *GeoInformatica* (2000) (also see http://www.lac.inpe.br/˜lorena/missae/index.html)