A Fast Algorithm for Point Labeling Problem

Sasanka Roy* Subhasis Bhattacharjee* Sandip Das* Subhas C. Nandy *

Abstract

In the map labeling problem, we are given a set $P = \{p_1, p_2, \ldots, p_n\}$ of point sites distributed on a 2D map. The label of a point p_i is an axis-parallel rectangle of specified size. The objective is to label maximum number of points on the map so that the placed labels are mutually non-overlapping. Here, we investigate a special class of map labeling problem where (i) the height of the label of each point is the same but its length may be different from the others, (ii) the label of a point p_i touches the point at one of its four corners and (iii) it does not obscure any other point in P. We describe an efficient heuristic algorithm for this problem which runs in $O(n\sqrt{n})$ time in the worst case. We run our algorithm as well as the algorithm proposed in [14] on the available benchmarks [13]. The results produced by our algorithm is same as that of [14] in most of the cases, and is one less in few cases. But the time taken by our algorithm is much less than [14].

1 Introduction

Labeling a point set is a classical problem in the geographic information systems, where the points represent cities on a map which need to be labeled with city names. It has a variety of practical applications as mentioned in The ACM Computational Geometry Impact Task Force report [2].

In general, the label placement problem includes positioning labels for area, line and point features on a 2D map. A good labeling algorithm has three basic requirements: (i) label of a site should touch the site at its boundary, (ii) it must not obscure the other sites on the map, and (iii) labels of two sites must not overlap. Many other aesthetic requirements for map labeling are listed in [6]. Given the basic requirements, two major types of problems are considered: (1) label as many sites as possible, and (2) find the largest possible size of the label such that all the sites can be labeled in a non-overlapping manner. In general, both of these problems are NP-hard [4]. In this paper, we shall consider a special case of the first variation of the point-site labeling problem.

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n points in the plane. For each point $p_i \in P$, we have a rectangular label r_i of specified size. The label of a point p_i must be placed parallel to the coordinate axes, and must contain p_i on one of the four corners of r_i. A label is said to be valid if it does not obscure any other point of P. A feasible solution is a family of axis-parallel rectangles (labels) $R = \{r_{1'}, r_{2'}, \ldots, r_{k'}\}$, each $i' \in \{1', 2', \ldots, k'\}$ is different and $r_{i'}$ is represented by a tuple $\{(p_{i'}, x_{i'}) \mid p_{i'} \in P$, $x_{i'} \in$ {top-left, top-right, bottom-left, bottom-right} and $r_{i'}$ is placed with $p_{i'}$ at its $x_{i'}$-th corner$\}$, such that the members in R are mutually non-overlapping. In other words, if we consider a graph whose nodes correspond to all possible valid labels, and an edge between a pair of nodes indicates that the corresponding two labels overlap, then a feasible solution correspond to an independent set of that graph. In subsequent discussions, we will refer this graph as *label graph*. The objective of the label placement problem is to find the maximum independent set of the label graph [1].

In [1], an $O(\log n)$-approximation algorithm is proposed for this problem which runs in $O(n\log n)$ time. An α-approximation algorithm produces a solution of size at least $\frac{\Delta}{\alpha}$, where Δ is the size of the optimal solution. In particular, if the labels are of the same height, a dynamic programming approach is adopted to get a $(1 + \frac{1}{k})$-approximation algorithm which runs in $O(n\log n + n^{2k-1})$ time [1]. This case is of particular importance since it models the label placement problem when all labels have the same font size. Note that, if each point is allowed to appear at its one specific corner (say top-left), then the maximum independent set of the label graph can easily be obtained in $O(n\log n)$ time [10]. A slightly generalized version of this problem, where a point is allowed to appear at either top-left or bottom-left corner of its label, becomes NP-complete. An $O(n\log n)$ time heuristic algorithm is proposed [10], but its performance is dominated by that of [14].

The point labeling with sliding labels is introduced in [7], where the point p_i can assume any position on the boundary of r_i. The problem has shown to be NP-hard, and using plane sweep paradigm, a 2-approximation algorithm has been presented whose time complexity is $O(n\log n)$. The label placement problem in the slider model has been extended for the map containing several polygonal obstacles, and the objective is to label a set of n point sites avoiding those obstacles [12]. The time complexity of this algorithm is $O((n + m)\log(n + m))$, where m and n are respectively the total number of vertices of all the polygons, and the number of point sites

*Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata - 700 108, India to be labeled. In [9], a decision-theoretic version of the map labeling problem is introduced where the sites are horizontal and vertical line segments. Each label has unit height and is as long as the segment it labels. The problem is to decide whether it is possible to label all the sites. The problem is transformed to the well known 2-satisfiability problem, and an algorithm for this decision problem is proposed which runs in $O(n^2)$ time. Later, in [11], the time complexity was improved to $O(n\log n)$. Good heuristics are proposed for labeling arbitrary line segments and polygonal areas [3]. In [14], a heuristic algorithm for the point labeling problem is proposed. It produces near-optimal solution in $O(n\log n + |E|)$ worst case time, where E is the set of edges in the *label graph*, and it may be $O(n^2)$ in the worst case.

Our proposed heuristic algorithm exploits the graph-theoretic properties of the label graph, it runs in $O(n\sqrt{n})$ time. Experimental results indicate that the solutions produced by our algorithm is same as that of [14] for most of the benchmark examples [13], and in a few cases is it less than [14] by only one. The merit of our algorithm is that it terminates in less than a minute for instances of size 1000, whereas the algorithm in [14] takes much time for similar examples. Thus, our algorithm can be utilized in practical environment for producing good quality solutions.

2 Algorithm

We consider the general problem where a site can appear at any of the four corners of its label. Thus, our objective is to compute the maximum independent set of the general label graph. We do it in two phases. In Phase I, we use four different configurations of valid labels as shown in Figure 1. In each configuration, a point can appear in three specified corners of its label. The rationale of choosing such configurations is that, we can easily identify cliques of the corresponding label graph during the plane sweep in horizontal/vertical direction, which leads to a near-optimal solution of the maximum independent set problem of its corresponding subgraph. For each configuration, we compute two such solution sets by executing plane sweep in horizontal and vertical directions. Thus, we have 8 different independent sets. In Phase II, we apply a merge pass on these 8 solution sets to compute the final solution, i.e., a large independent subset of nodes in the general label graph. It is observed that, in all the cases the solution is almost of same size as that of the best solution of the problem proposed in [14].

2.1 Phase I

Consider the configuration in Figure 1(a). For each point $p_i \in P$, draw the three possible labels, and for each of them test for its *validity*. Next, sort the valid

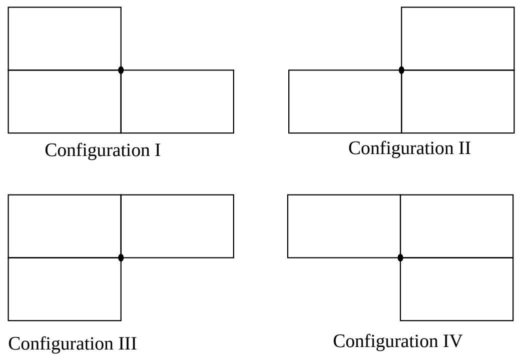

Figure 1: Possible configurations of valid labels

labels with respect to the y-coordinates of their horizontal boundaries, and perform a horizontal line sweep from top to bottom for generating a set of non-overlapping labels, called the *independent set*, and is denoted by h_list_1. Note that, if the top-left label for a point p_i is selected in the independent set, then the other two labels will not appear in the independent set. However, if this label can not be chosen for the presence of a label of some other point, then one of the remaining two labels of p_i are tested for inclusion in the independent set. We also compute another independent set, called v_list_1, by sweeping a vertical line from right to left. This needs sorting of the vertical boundaries of the valid labels with respect to their x-coordinate.

Similarly, for the configurations in Figures 1(b), 1(c) and 1(d), horizontal line sweeps are executed from bottom-to-top, top-to-bottom and bottom-to-top respectively, and the name of the corresponding independent sets are h_list_2, h_list_3 and h_list_4 respectively. The directions of the vertical line sweep for Figures 1(b), 1(c) and 1(d) are right-to-left, left-to-right and left-to-right respectively. The corresponding independent sets are v_list_2, v_list_3 and v_list_4 respectively.

2.2 Phase II

In this phase, our objective is to compute an estimate of the maximum independent set of the label graph by merging the independent sets $h\_list_1, \ldots, h\_list_4$, $v\_list_1, \ldots, v\_list_4$ obtained in Phase I.

Consider two independent sets, say U and W of the same label graph, and obtain a graph $G_B(U, W; E)$ as follows: put edges between nodes in U and W such that for an edge $(u, w) \in E$, we have $u \in U$, $w \in W$ and the labels corresponding to u and w overlap. Needless to say, the graph G_B is bipartite since an edge can not exists between a pair of nodes in U (resp. W). The size of the maximum independent set of G_B is greater than or equal to $max(|U|, |W|)$. We can use the algorithm in [8] to obtain a maximum independent set of a bipartite graph in $O(|E|)$ if the maximum matching of the graph

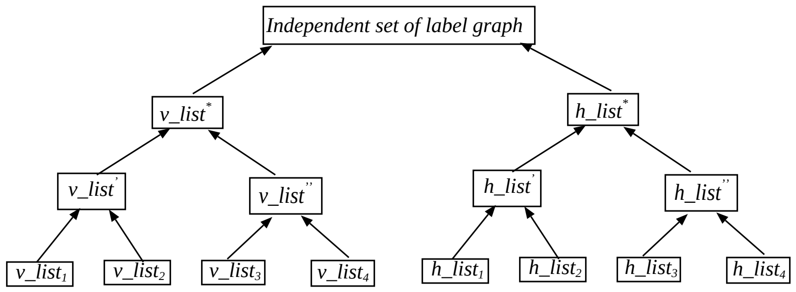

Figure 2: Hierarchical merging in Phase 2

G_B is given. The maximum matching of a bipartite graph can be obtained in $O(|E|\sqrt{|U|+|W|})$ time [5].

Lemma 1 *If U and W corresponds to two lists in $\{h\_list_1, h\_list_2, h\_list_3, h\_list_4, v\_list_1, v\_list_2, v\_list_3, v\_list_4\}$, then the G_B is a planar graph.*

Proof. We propose a construction technique of the planar embedding of the graph G_B to prove the result.

Consider a label $\alpha \in U$. Let it intersects with $\beta_1, \beta_2, \ldots, \beta_k \in W$. Since the graph G_B is bipartite, the regions $\{\alpha \cap \beta_i, i = 1, 2, \ldots, k\}$ and $(\alpha - \cup_{i=1}^{k}\beta_i)$ are mutually non-intersecting. As the labels $\{\beta_i, i = 1, 2, \ldots, k\}$ are not intersecting even at their boundaries, the region $(\alpha - \cup_{i=1}^{k}\beta_i)$ is non-empty.

Again, (i) if a label β_i spans both the horizontal boundaries of α then it contains exactly one of the vertical boundaries of α (since α is a valid label and of same height with β_i), and (ii) if β_i contains portions of two vertical boundaries of α then one of the horizontal boundaries of α must lie inside β_i. Thus, $(\alpha - \cup_{i=1}^{k}\beta_i)$ is a connected region.

Thus, we place the node (point) corresponding to α, called v_α, inside $(\alpha - \cup_{i=1}^{k}\beta_i)$. Note that, we can reach from v_α to a point inside $\alpha \cap \beta_i$ by a path $\pi(\alpha, \beta_i)$ which completely stays inside α and does not pass through any of the labels $\beta_j, j = 1, 2, \ldots, k$ except β_i. In other words, the paths $\pi(\alpha, \beta_i), i = 1, 2, \ldots, k$ are non-intersecting.

Thus in the embedding of G_B, we can place a node v_α for each member $\alpha \in U$. Similarly, for each member in W, we place a node. If the labels corresponding to a node $\alpha \in U$ intersects with $\beta \in W$, we can draw the edge (v_α, v_β) by concatenating the paths $\pi(\alpha, \beta)$ and $\pi(\beta, \alpha)$. This edge will not intersect any other labels except the labels corresponding to α and β. This proves that any two edges in the graph G_B do not intersect. $\square$

Let us consider a permutation of $\{h\_list_i, i = 1, 2, 3, 4\}$ and $\{v\_list_i, i = 1, 2, 3, 4\}$, and hierarchically merge these eight independent sets as follows to get a larger independent set of the label graph.

The first two members in the permutation of h_lists are named as U and W, and the independent set of G_B is computed. Name it h_list'. Similarly h_list'' is computed by merging the last two members of the permutation of h_lists. Finally, h_list^* is computed by merging h_list' and h_list''. Similarly, v_list', v_list'' and then v_list^* is computed. Finally, h_list^* and v_list^* are merged to get the independent set of the label graph (see Figure 2 for the demonstration).

Note that, there may be three different permutations of h_lists which may generate different h_list^*s. Same thing holds for the v_lists also. Thus, we need to inspect 9 different (h_list^*, v_list^*) pairs. Finally, we report one of these 9 solutions which is having maximum cardinality.

2.3 Complexity analysis

Theorem 2 *The time complexity of our algorithm is $O(n\sqrt{n})$ in the worst case.*

Proof. The generation of eight lists of independent sets in Phase I needs $O(n\log n)$ time. Each merging phase can be executed in $O(n\sqrt{n})$ time, since the number of edges in G_B is $O(n)$ (by Lemma 1). For one particular permutation of h_lists and v_lists, we need to execute the merge algorithm seven times (see Figure 2). As we need to consider nine different permutations of h_lists and v_lists, the result follows. $\square$

3 Experimental results

We have executed this algorithm on many randomly generated examples and on all the benchmark examples available in [13]. The experimental results appear in Table I. In most of the cases, the results obtained by our algorithm are same as that of [14]. In very few cases, our algorithm produces a solution which is 1 or 2 less than that of [14]. For some example, we could obtain the optimum solution, which are also presented in the table. The run time requirement of our algorithm is also mentioned in the table. We like to mention that, the running time of (our implementation of) the algorithm in [14] needs very high time in comparison

Table 1: Experimental results

Examples	No. of sites	height	Opt. soln.	soln. in [14]	Our Algorithm	
					soln.	time (in sec.)
Tourist shops in Berlin	357	4	***	180	179	0.3317
		5	***	152	151	0.7868
German railway stations	366	4	347	347	346	0.2250
		5	***	332	332	0.1763
American cities	1041	4	1041	1041	1041	0.5000
		5	1041	1041	1041	0.5000
Drill holes in Munich	250	604	***	249	249	0.1336
	3000	521	3000	3000	2998	3.3842
	19461	5000	***	***	11049	7.1452

*** indicates optimum solution can not be found for that example.

to that of ours. There are many other examples in [13] for which our algorithm can give solution very quickly but the algorithm of [14] can not terminate in one day.

Acknowledgment: We are thankful to Dr. Alexander Wolff for providing us the benchmark examples.

References

[1] P. K. Agarwal, M. van Kreveld and S. Suri, *Label placement by maximum independent set in rectangles*, Computational Geometry: Theory and Applications, vol. 11, pp. 209-218, 1998.

[2] B. Chazelle et at, Application challenges to computational geometry: CG impact task force report, http://www.cs.princeton.edu/čhazelle/taskforce /CGreport.ps, 1996.

[3] S. Edmondson, J. Christensen, J. Marks, and S. Shieber, *A general cartographic labeling algorithm*, Cartographica, vol. 33, no. 4, pp. 13-23, 1997.

[4] M. Formann and F. Wagner, *A packing problem with applications to lettering of maps*, Proc. 7th. Annual ACM Symp. on Computational Geometry, pp. 281-288, 1991.

[5] J. E. Hopcroft and R. E. Karp, *An $n^{\frac{5}{2}}$ algorithm for maximum matching in bipartite graphs*, SIAM J. Computing, vol. 2, pp. 225-231, 1973.

[6] E. Imhof, *Positioning names on maps*, The American Cartographer, vol. 2, no. 2, pp. 128-144, 1975.

[7] M. van Kreveld, T. Strijk and A. Wolff, *Point labeling with sliding labels*, Computational Geometry: Theory and Applications, vol. 13, pp. 21-47, 1999.

[8] T. Kashiwabara, S. Masuda, K. Nakajima and T. Fujisawa, *Generation of maximum independent sets of a bipartite graph and maximum cliques of a circular-arc graph*, Journal of Algorithms, vol. 13, pp. 161-174, 1992.

[9] C. K. Poon, B. Zhu and F. Chin, *A polynomial time solution for labeling a rectilinear map*, Proc. 13th. ACM Symp. on Computational Geometry, pp. 451-453, 1997.

[10] S. Roy, P P. Goswami, S. Das, S. C. Nandy, *Optimal algorithm for a special point-labeling problem*, Information Processing Letters, vol. 89(2), pp. 91-98, 2004.

[11] T. Strijk and M. van Kreveld, *Labeling a rectilinear map more efficiently*, Information Processing Letters, vol. 69, pp. 25-30, 1999.

[12] T. Strijk and M. van Kreveld, *Practical extension of point labeling in the slider model*, 7th. Int. Symp. on Advances in Geographical Information Systems, (ACM-GIS '99), pp. 47-52, 1999.

[13] A. Wolff, General Map Labeling Webpage, http://www.math-inf.uni-greifswald.de/map-labeling/general/.

[14] F. Wagner, A. Wolff, V. Kapoor and T. Strijk, *Three rules suffice for good label placement*, Algorithmica, vol. 30, pp. 334-349, 2001.