Available online at www.sciencedirect.com

ScienceDirect

Information Processing Letters 105 (2008) 108–113

Information Processing Letters

www.elsevier.com/locate/ipl

# Optimal point removal in closed-2PM labeling [☆]

Farshad Rostamabadi [a,b,*], Iman Sadeghi [a], Mohammad Ghodsi [a,b], Ramtin Khosravi [c]

[a] *Computer Engineering Department, Sharif University of Technology, Tehran, Iran*
[b] *School of Computer Science, Institute for Studies in Fundamental Sciences, Tehran, Iran*
[c] *Electrical and Computer Engineering Department, University of Tehran, Tehran, Iran*

Received 17 July 2006

Available online 25 September 2007

Communicated by F. Dehne

**Abstract**

An optimal labeling where labels are disjoint axis-parallel equal-size squares is called 2PM labeling if the labels have maximum length each attached to its corresponding point on the middle of one of its horizontal edges. In a closed-2PM labeling, no two edges of labels containing points should intersect. Removing one point and its label, makes free room for its adjacent labels and may cause a global label expansion. In this paper, we construct several data structures in the preprocessing stage, so that any point removal event is handled efficiently. We present an algorithm which decides in $O(\lg n)$ amortized time whether a label removal leads to label expansion in which case a new optimal labeling for the remaining points is generated in $O(n)$ amortized time.
© 2007 Elsevier B.V. All rights reserved.

*Keywords:* Computational complexity; Map labeling

## 1. Introduction

Optimal labeling of a set of points is generally a NP-Hard problem, but with some restrictions, like ones in the well known 1P and 2P models in point labeling [1], it can be solved in polynomial time. Also, the special cases of elastic labeling introduced in [2] where labels should have an edge on the positive part of $x$ and $y$ axis can be solved in polynomial time with a dynamic programming technique. In another paper, a special case of the line labeling where all lines are orthogonal and each label has only three candidates, are solved in polynomial time by a reduction to the 2SAT problem [3,8].

As other polynomially solvable problems in map labeling, we consider two specific labeling models, called 2PM and 4PM, where the labels for all points are equal-size axis-parallel squares with maximum possible size. The main restriction is that, each label can be attached to its point only on the middle of one of its horizontal or vertical edges. In 2PM model, each point has two choices: top or bottom edges (or similarly left or right edges), but in 4PM, each point chooses one of either top/bottom or left/right edges, and this is known in advance.

The problems of dynamic insertion of point-obstacles or new labels into an existing labeling and a fast reconstruction of a new optimal labeling (with possibly shrunk labels) have been considered before. Relabel-

[☆] Supported in part by Institute for Theoretical Physics and Mathematics (IPM) under grant number CS1385-2-01.

[*] Corresponding author at: Computer Engineering Department, Sharif University of Technology, Tehran, Iran.

*E-mail address:* farshad@parsebiz.com (F. Rostamabadi).

ing in 4PM to avoid a single obstacle was considered in [4] and a simpler algorithm in 2PM was later proposed in [5]. The same authors consider the problem of incremental labeling for 4PM model in [7] and a simpler algorithm for the 2PM case was presented in [6]. The latter algorithms do: (a) insert a label without shrinking the labels in $O(\lg n)$ time and, (b) relabel all points with smaller optimal labels in linear time, if part (a) is not possible.

In this paper, we study the problem of removing a label from a given optimal labeling. We perform some preprocessing in $O(n \lg n)$ time and $O(n)$ space so that we can decide in $O(\lg n)$ amortized time whether such a removal leads to a new optimal labeling with expanded labels. We can then use [6] to relabel all points in $O(n)$ amortized time if needed. We will see that our removal algorithm cannot easily be combined with our insertion results, so an efficient and fully dynamic algorithm to deal with both insertion and removal is our current challenge.

## 2. Problem definition and the method outline

A 2PM labeling $\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_n\}$ of a point set $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ is a set of disjoint square axis-parallel labels with side length $\sigma$ positioned on the plane such that each label $\ell_i$ is attached to $p_i$ on the middle of one of its horizontal edges [4,5]. The edge of $\ell_i$ containing $p_i$ is denoted by $ref(\ell_i)$ and is called its *reference edge*, and its opposite is denoted by $opp(\ell_i)$. Also, each of its two other edges is called its *side edge*. A label $\ell_i$ can be flipped only on its reference edge, the resulting label is denoted as $f(\ell_i)$. A *Closed-2PM labeling* is a variation of 2PM labeling where there is no pair of overlapping reference edges. A closed-2PM labeling with the maximum length is called an *optimal labeling*.

Given a point set $\mathcal{P}$ and its optimal labeling $\mathcal{L}$, suppose a point $p_i$ with its $\ell_i$ is removed from $\mathcal{L}$. If $\mathcal{L} - \{\ell_i\}$ is not optimal, we should compute a new optimal labeling for $\mathcal{P} - \{p_i\}$. The problem is to preprocess the input, so that upon each removal of a point like $p_i$ we can efficiently determine if $\mathcal{L} - \{\ell_i\}$ is still optimal. If not, we use the algorithm presented in [6] to relabel $\mathcal{P} - \{p_i\}$ optimally with enlarged labels.

Using a naïve approach, the check for the optimality of a labeling can be done in $O(n)$. To improve this time, we compute and maintain a data structure called *conflict graph* $\mathcal{G}$ (to be defined later). Likewise, computing a new labeling from scratch takes $O(n \lg n)$ time using a 2SAT approach [1,7]. The algorithm in [6] uses another data structure called *adjacency graph* $\mathcal{H}$ (originally defined in [7]), to improve this running time to $O(n)$. The adjacency graph can be constructed in $O(n \lg n)$ time and $O(n)$ space and updated in $O(\lg n)$ time per each removal.

The main steps of the algorithm are as follows. In this paper, we concentrate on the optimality check step and on the construction and maintenance of the conflict graph $\mathcal{G}$. Other steps have been worked out before.

*Preprocessing*: Compute $\mathcal{G}$ and $\mathcal{H}$.
*Point Removal*: Upon removal of a point $p_i$:
- *Optimality Check*: Using graph $\mathcal{G}$, test if $\mathcal{L} - \{\ell_i\}$ is still optimal.
- *Update*: Remove $p_i$ from $\mathcal{P}$ and update the two graphs $\mathcal{G}$ and $\mathcal{H}$.
- *Relabel*: In case the optimality check was negative, compute a new optimal labeling.

## 3. The algorithm

Consider the labeling $L' = \mathcal{L} - \{\ell_i\}$ that is obtained after removal $\ell_i$ from the original optimal labeling $\mathcal{L}$. If there are no two labels with touching boundaries, then $L'$ is obviously not optimal. Otherwise, to see if $L'$ is still optimal, we should check if we can flip some labels such that there are no more pairs of touching labels. We consider each touching pair $(\ell_i, \ell_j)$ in turn and see if $\ell_i$ or $\ell_j$ (possibly with some other labels) can be flipped to separate $\ell_i$ and $\ell_j$. If the touching labels cannot be separated, then $L'$ is optimal.

To flip a label $\ell_i$ and still having non-intersecting labels, we should flip every other label that intersects $f(\ell_i)$ too. The flips are propagated recursively. The end to this recursion happens with two possibilities: (1) the flips end with no intersecting labels, and (2) when some $\ell_k$ is to flip and $f(\ell_k)$ intersects with both $\ell_l$ and $f(\ell_l)$, for some $\ell_l$. So, flipping of $\ell_i$ is not possible in the second case.

There are two special situations in which it is possible to flip some labels, yet the length of the labeling is not increased. First, it is possible to have two pairs of touching labels $(\ell_i, \ell_j)$ and $(\ell_k, \ell_l)$, such that each pair can be separated individually, but not at the same time (as illustrated in Fig. 1(a)). Second, it is possible to have a pair of touching labels $(\ell_i, \ell_j)$ that can be separated, but applying the flips to separate these labels creates another pair of touching labels. It can be easily seen that separating the new pair causes $(\ell_i, \ell_j)$ to touch again (as illustrated in Fig. 1(b)).

The process of optimality check as described above (i.e., separating each pair of touching labels in turn by recursive flips) is not efficient. The problem we are facing is to efficiently determine which touching pairs can

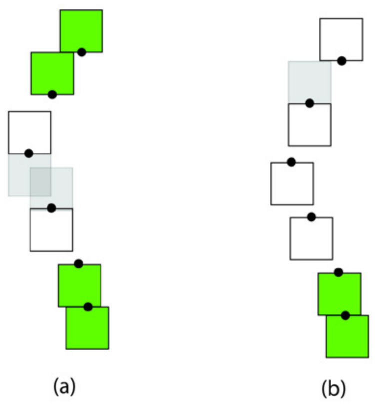

Fig. 1. Two special cases while removing the touching pairs of labels (green labels are touching, and gray labels are flipped version of original labels): (a) Only one pair of the touching labels can be removed; (b) Removing the touching pair creates another pair of touching labels.[1]

be separated using the free room previously occupied by the label just removed. To solve this, we use the conflict graph introduced in [4,7]. The conflict graph $\mathcal{G} = (\mathcal{P}, \mathcal{F} \cup \mathcal{B})$, is a multi-graph with $\mathcal{P}$ as its vertices. A directed edge $(p_i, p_j)$ belongs to $\mathcal{F}$ if and only if $f(\ell_i)$ intersects $\ell_j$, but does not intersect $f(\ell_j)$. An undirected edge $(p_i, p_j)$ belongs to $\mathcal{B}$ if and only if $f(\ell_i)$ intersects $f(\ell_j)$ or touches $\ell_j$. Intuitively, having an edge $(p_i, p_j)$ in $\mathcal{F}$ means $\ell_i$ cannot be flipped unless $\ell_j$ is flipped too. The edges in $B$ help us find the end of the flipping sequences. We sometimes slightly abuse the notation and use the labels corresponding to the points as the vertices of $\mathcal{G}$.

The conflict graph is constructed in the preprocessing phase, when no point is removed yet. Note that it is possible for the initial labeling to contain some touching pairs that can be separated successfully. Prior to constructing the graph, we separate these touching pairs one by one until no pairs can be separated successfully (without generating a new touch). We call each remaining touching pair, a *constraint pair*.

We can observe that a constraint pair $(\ell_i, \ell_j)$, can only be of one of the following three categories (as illustrated in Fig. 2):

*Type* 1: Only one endpoint of $\mathit{ref}(\ell_i)$ coincides with one endpoint of $\mathit{ref}(\ell_j)$.

*Type* 2: $\mathit{ref}(\ell_i)$ touches $\mathit{opp}(\ell_j)$ or $\mathit{opp}(\ell_i)$ touches $\mathit{ref}(\ell_j)$.

*Type* 3: $\mathit{opp}(\ell_i)$ touches $\mathit{opp}(\ell_j)$ or two side edges are touching.

Suppose that we have removed a point with its label and have a constraint pair $(\ell_i, \ell_j)$. If flipping $\ell_i$ removes the constraint, we call $\ell_i$ a *relaxing label*. The same rule applies to $\ell_j$ as well. For a type 1 constraint pair, none of the labels are relaxing, since flipping labels cannot relax the constraint. For a type 2 constraint pair, assuming $\mathit{ref}(\ell_i)$ touches $\mathit{opp}(\ell_j)$, $\ell_j$ is relaxing. $\ell_i$ is not relaxing in this case since $f(\ell_i)$ touches $\mathit{ref}(\ell_j)$. Finally, for a type 3 constraint pair, either one label or both of them are relaxing.

If a relaxing label $\ell_i$ is to flip, we will have a sequence of recursive flips. Since $\ell_i$ is a part of a constraint pair, we fail doing all these flips. But, not all labels considered during this process are failed to flip. We mark those that fail as *red*. Any label that is not red is marked as *green*.

**Definition 1.** A path $\pi = (\ell_{i_1}, \ell_{i_2}, \ldots, \ell_{i_m})$ of labels is a *complete red path* if,

1. $\ell_{i_1}$ is relaxing,
2. $(\ell_{i_j}, \ell_{i_{j+1}}) \in \mathcal{F}$ for all $1 \leqslant j < m$, and
3. there is a label $\ell_k$ where
   (a) $f(\ell_{i_m})$ intersects both $\ell_k$ and $f(\ell_k)$ (i.e. $(\ell_{i_m}, \ell_k) \in \mathcal{B}$), or
   (b) $f(\ell_{i_m})$ intersects $f(\ell_k)$ and there is a path of edges in $\mathcal{F}$ from a relaxing label to $\ell_k$.

We mark $\ell_{i_1}, \ell_{i_2}, \ldots, \ell_{i_m}$ as red labels and denote any sub-path of $\pi$ as a *red path*.

The item 3 in the above definition of path $\pi$ specifies the condition where a sequence of flips finally fails.

Having the path $\pi$ as defined above, suppose a label $\ell_{i_d}$ $(1 \leqslant d \leqslant m)$ is to be removed. Since the removal of $\ell_{i_d}$ disconnects $\pi$, the remaining labels on this path can be turned into green, unless they also belong to some other red path(s). To do this, we successively apply the following rules and try to turn all red labels into green.

**Rule 1.** If a red label $\ell_x$ has no outgoing edge in $\mathcal{F}$ to a red label, and has no connecting edge in $\mathcal{B}$ to a red label, $\ell_x$ is turned into green.

To describe this rule, suppose $\ell_y$ is red. Having the edge $(\ell_x, \ell_y)$ in $\mathcal{F}$ means that flipping $\ell_x$ requires $\ell_y$ to flip which is impossible. For the second case and assuming that $\ell_x$ and $\ell_y$ are red and the edge $(\ell_x, \ell_y)$ in $\mathcal{B}$. This means that there is not enough room for both labels

[1] For interpretation of the references to color, the reader is referred to the web version of this article.

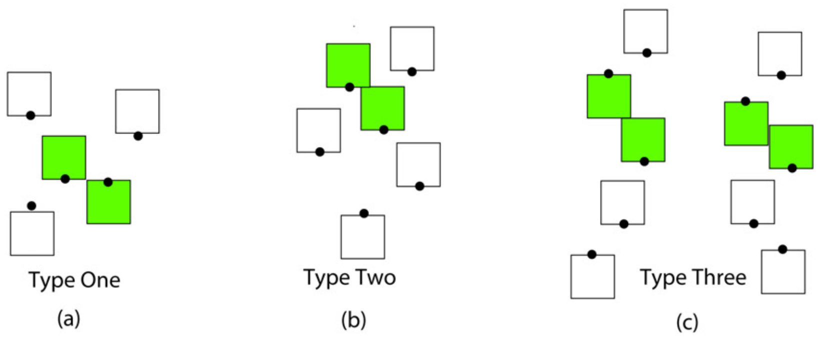

Fig. 2. Three different types of constraint pairs: (a) Touching reference edges; (b) A reference edge touches the opposite edge; (c) Touching opposite edges (left) and touching side edges (right).

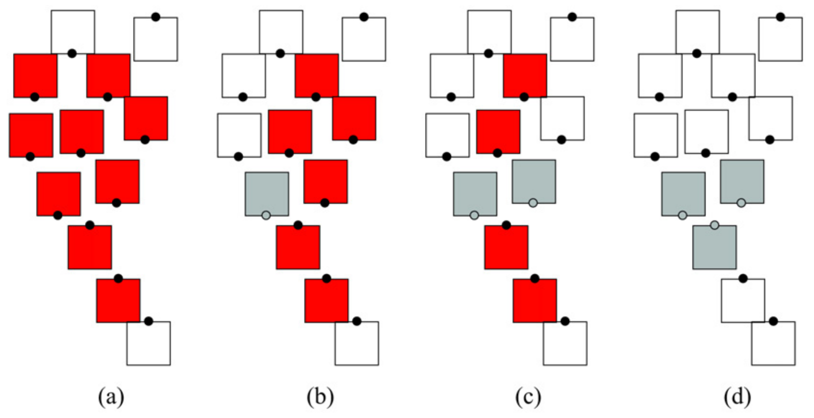

Fig. 3. A step by step example of color changes in a sample labeling (green labels are shown as white). In the left, a sample labeling with initial label colors is shown. At each step, a label, shown in gray, is deleted and label colors are adjusted. Finally there is no more red labels hence the points should be relabeled.

to flip at once. Note that only one of these two labels can be turned into green. But, this would not turn the other into green, we leave both as red.

**Rule 2.** If a non-relaxing red label $\ell_x$ has no incoming edge in $\mathcal{F}$ from another red label, it is turned into green.

This rule implies that, having no incoming edge from another red label in $\mathcal{F}$, means that nobody forces $\ell_x$ to flip, so it can safely turn into green.

**Rule 3.** If a red relaxing label $\ell_x$ which is a part of a constraint pair of type 3 turns into green, the other label is turned into green (if it is not already green).

To make a procedural view of color updates, observe that removing and changing the color of a label may only affect the color of its neighbors in $\mathcal{G}$. So, we start by the neighbors of the label just removed, and recursively apply the above rules to the neighbors of the labels just turned into green.

In case all labels are turned into green, we should relabel the whole map with enlarged label length, and re-color all labels to prepare for the next removal. If there exist some red labels, the current labeling is optimal and this is proved in the following section. Fig. 3 illustrates how label colors are changes, according to our algorithm, for an arbitrary input. After deleting three labels, the remaining points of Fig. 3 should be relabeled and recolored since there is no more red labels.

## 4. Proof of correctness and analysis

**Lemma 1.** *After a point removal, assume that no more red label be turned into green. Then each red label belongs to at least one complete red path.*

**Proof.** Consider a red label $\ell_x$ that does not belong to any complete red path. Two cases may happen: either we fail reaching a red label $\ell_y$ from $\ell_x$ such that $\ell_y$ has an edge in $\mathcal{B}$ to another red label, or there is no red path from a relaxing label to $\ell_x$. In the first case, consider a longest red path from $\ell_x$ to some other label $\ell_z$, in which case, Rule 1 changes the color of $\ell_z$ into green which is a contradiction. In the second case, consider a longest red path from some label $\ell_z$ to $\ell_x$. In this case, $\ell_z$ is not relaxing and has no incoming edge in $\mathcal{F}$ from a red label. So, the second rules is applicable, which is a contradiction. □

The following lemma is easy to prove based on Lemma 1 and Rule 3.

**Lemma 2.** *After a point removal, assume that no more red label be turned into green. If a relaxing label $\ell_x$ is green, then either $\ell_x$ can be successfully flipped, or $\ell_x$ is a part of a constraint of type 3 and the other label in the constraint pair can be successfully flipped.*

**Proof.** The label $\ell_x$ is made green by either Rule 1 or Rule 3. In the first case, there is no red path starting from $\ell_x$, so it can be successfully flipped. Rule 3 yields to the second case stated in the lemma. □

Finally, the following lemma concludes the correctness of the algorithm.

**Lemma 3.** *After a point removal, assume that no more red label be turned into green. The length of the labeling can be increased if and only if no red label exists.*

**Proof.** If a red label exists, according to Lemma 1, there is a complete red path which means there is a constraint pair that cannot be resolved. If there is no red label, then any constraint pair contains green labels and can be resolved according to Lemma 2. □

To analyze the running time of the algorithm, assume that a simple algorithm applies the three rules recursively to the neighbors of the label just removed. The label $\ell_y$ is a neighbor of $\ell_x$ if $\ell_x$ and $\ell_y$ are connected in $\mathcal{F}$ or $\mathcal{B}$. Then, the algorithm applies the rules to the neighbors of all labels turned into green. We use the potential method to show that the time of color changes for each point removal, is amortized $O(1)$. This should be added to the cost of updating $\mathcal{H}$ which is always $O(\lg n)$ [6]. So, each point removal needs $O(\lg n)$ amortized time. Define the function $\phi(\mathcal{L}) = 9r(\mathcal{L})$ as the potential function of $\mathcal{L}$ where $r(\mathcal{L})$ is the number of red labels in $\mathcal{L}$. Since each label has at most eight neighbors, each label is considered at most eight times when its neighbor turns into green and once when the label itself turns into green. So, the actual cost of visiting and assigning new colors to red labels and their neighbors are covered by the potential function.

Thus, the main result of the paper can be summarized in the following theorem:

**Theorem 1.** *Having an optimal labeling $\mathcal{L}$ of a set $P$ of $n$ points in closed-2PM labeling, and using an $O(n \lg n)$ time and $O(n)$ space preprocessing, assume that an arbitrary event point $p_i \in P$ is removed from the map. We can decide in $O(\lg n)$ amortized time whether $\mathcal{L} - \{\ell_i\}$ is optimal. If not, a new optimal labeling with enlarged labels can be constructed in $O(n)$ amortized time.*

## 5. The relabeling algorithm

For the sake of completeness, we briefly mention the main ideas of the relabeling algorithm [6] of a set of points in closed-2PM model.

The main idea is that in every optimal closed-2PM labeling, the top most label can be placed above its point. This idea, generates a sweep line based algorithm, which moves a sweep line upward, and stops at each point of $\mathcal{P}$. At each stop, all points not above the sweep line are labeled optimally. In other words, in the very first steps, there are a few points labeled by large labels and as the sweep line moves upward, more points are involved and the labels are shrunk.

Assume that the sweep line is going to label $p_i$, and the $\ell_i$ intersects with an existing label, like $\ell_j$. Authors showed the following properties: (a) $\ell_j$ is forced to flip down, which may generate a label shrink, that can be computed from a set of stored weights at each label in previous steps, (b) no label flips twice, and (c) there is no need to update previously stored weights at each label. By combining all these properties together and spending $O(n \lg n)$ time and $O(n)$ space for preprocessing, the sweep line algorithm can assign optimal labels to a set of points in $O(n)$ time.

## 6. Conclusion

In this paper, we proposed an algorithm to preprocess an optimal labeling of a set of $n$ points in $O(n \lg n)$ time and $O(n)$ space, so that after removal of any point, one can efficiently determine whether the remaining labels form an optimal labeling in $O(\lg n)$ amortized time. If the labeling is not optimal, a relabeling algorithm can be used to construct an optimal labeling in $O(n)$ amortized time. Using this result, we can start with an optimal labeling and remove an arbitrary number of points, and relabel the point set only when the length of the optimal labeling can be increased.

There are two possible directions to extend the algorithm. One is to provide a method for the similar problem in 4PM labeling. Applying the method used in this paper directly to the 4PM case is not easy, since there are more possible cases for constraint pairs in 4PM. Another problem is to consider combination of point insertion and removals. This extension is not straightforward and is our current undergoing challenge.

## References

[1] S. Doddi, M.V. Marathe, A. Mirzaian, B.M.E. Moret, B. Zhu, Map labeling and its generalizations, in: Proc. 8th ACM–SIAM Symposium on Discrete Algorithms (SODA'97), 4–7 January 1997, pp. 148–157.

[2] C. Iturriaga, Map labeling problems, PhD thesis, University of Waterloo, 1999.

[3] C.K. Poon, B. Zhu, F. Chin, A polynomial time solution for labeling a rectilinear map, Information Processing Letters 65 (4) (1998) 201–207.

[4] F. Rostamabadi, M. Ghodsi, A fast algorithm for updating a labeling to avoid a moving point, in: Proceeding of the 16th Canadian Conference on Computational Geometry (CCCG'04), 2004, pp. 204–208.

[5] F. Rostamabadi, M. Ghodsi, Label updating in 2pm to avoid a moving point, in: The 21st European Workshop on Computational Geometry (EWCG'05), 2005, pp. 131–134.

[6] F. Rostamabadi, M. Ghodsi, Incremental labeling in 2pm model, in: The 11th International CSI Computer Conference (CSICC'06), 2006, pp. 9–13.

[7] F. Rostamabadi, M. Ghodsi, Label updating to avoid point-shaped obstacles in fixed model, Theoretical Computer Science 369 (1–3) (2006) 197–210.

[8] T. Strijk, M. van Kreveld, Labeling a rectilinear map more efficiently, Information Processing Letters 69 (1) (1999) 25–30.