Available online at www.sciencedirect.com

ScienceDirect

Theoretical Computer Science 369 (2006) 197–210

Theoretical Computer Science

www.elsevier.com/locate/tcs

Label updating to avoid point-shaped obstacles in fixed model

Farshad Rostamabadi[a, b, *], Mohammad Ghodsi[a, b]

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

Received 18 December 2004; received in revised form 5 August 2006; accepted 17 August 2006

Communicated by S. Sen

Abstract

In this paper, we present efficient algorithms for updating the labeling of a set of n points after the presence of a random obstacle that appears on the map repeatedly. We update the labeling so that the given obstacle does not appear in any of the labels, the new labeling is valid, and the labels are as large as possible (called the optimal labeling). Each point is assumed to have an axis-parallel, square-shaped label of unit size, attached exclusively to that point in the middle of one of its edges. We consider two models: (1) the 2PM model, where each label is attached to its feature only on the middle of one of its horizontal edges, and (2) the r4PM model, where each label is attached to its feature on the middle of either one of its horizontal or vertical edges (known in advance). We assume that a sequence of point-shaped obstacles appear on the map on random locations. Three settings are considered for the behavior of the obstacle: (1) the obstacle is removed afterwards, (2) it remains on the map, and (3) it receives a similar label and remains on the map. Only two operations are permitted on the labels: flipping one or more labels, and/or resizing all labels. In the first setting, we suggest a data structure of $O(n)$ space and $O(n \lg n)$ time in the 2PM model, and of $O(n^2)$ time in the r4PM model, so that the updated labeling can be constructed for any obstacle position in $O(\lg n + k)$ time, where k is the minimum number of operations needed. For the second and third problems, we suggest an $O(n)$ space and $O(n \lg n)$ time data structure that can place each obstacle (possibly with a label) on the map in $O(\lg n + k)$ time, if k label flips are sufficient to make room to place the new obstacle. Otherwise, two $O(n)$ time algorithms are suggested when a relabeling of all points is required.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Computational geometry; Map labeling; Label updating

1. Introduction and problem definitions

Automated label placement is an important problem in map generation and geographical information systems. This is to attach one or more labels (regularly in text) to each feature of the map, which may be a point, a line, a curve, or a region. The point feature label placement has received considerable attention. There are two basic requirements of any labeling; labels should be pairwise disjoint, and each label should have a common point with its feature [8]. Other variations of this problem let the features receive more than one labels [2], or use specific shapes for the labels [1,6,7,18]. There are also two labeling models, fixed and slider model. In the former, some fixed positions of each label are given as the possible locations for feature placements [8], and in the latter, the labels can be placed at any position

* Corresponding author. Computer Engineering Department, Sharif University of Technology, Tehran, Iran. Tel.: +98 (21) 6005616.
E-mail addresses: farshad@mehr.sharif.edu (F. Rostamabadi), ghodsi@sharif.edu (M. Ghodsi).

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.08.038

while touching the features [7,17,5]. The optimal labeling of a set of points (with different optimality measures) is generally an NP-Complete problem, but with some restrictions, it can be solved in polynomial time, like the special point labeling in 1P and 2P models [14], the elastic labeling introduced in [4], and the line labeling in special cases [10,16].

In this paper, we consider the point labeling problems in the fixed model, where each of the point-shaped features receives a unit-length axis-parallel square-shaped label. More formally, the input is assumed to be a labeling composed of points $\mathcal{P} = \{p_1, p_2, \dots, p_n\}$ and unit-length labels $\mathcal{L} = \{\ell_1, \ell_2, \dots, \ell_n\}$, where ℓ_i is attached to p_i on the mid-point of one of its edges. We consider two labeling models: (1) the 2PM model, where one label is attached to each feature on the middle of one of its horizontal (or similarly vertical) edges, and (2) the r4PM model, a restricted version of the 4PM model, where a label can be attached to its feature point only on the middle of either one of its horizontal or vertical edges. [1] The restriction here is that this choice is given in advance for each point p, which is called its *label direction* and denoted as $d(p)$. A point with horizontal (resp. vertical) label direction should be attached to the middle of one of its horizontal (resp. vertical) edges of its label. A labeling that is valid with the largest possible label size is called *the optimal labeling.*

We are interested in fast algorithms to update an optimal labeling when one or more point-shaped obstacles appear on the map at random locations, so that the new optimal labeling does not contain the obstacles.

This problem is considered in the following three settings:

POAR *Point-obstacle-arrive-then-removed*, where one obstacle appears randomly on the map, stays for a while and is then removed. This may happen when the latest position of a moving obstacle is repeatedly reported (like in a radar aviation system or in graphic games). When the obstacle is present, a new optimal labeling is to be constructed that avoids the obstacle. The labeling goes back to its original state after the obstacle is removed. This may happen quite often, thus a fast updating algorithm is needed.

POAS *Point-obstacle-arrive-and-stay*, where obstacles are introduced incrementally and remain on the map forever. We should update the labeling after each obstacle arrival.

LOAS *Labeled-obstacle-arrive-and-stay*, this is the same as POAS, but each obstacle should also receive a label of the same length and in the same model as others. The labeled obstacle remains on the map forever. This problem can also be thought of as an *incremental labeling* where points are introduced incrementally and all points should have optimal labeling after each new arrival. In the r4PM model, each new point has its label direction, known in advance.

The updated labeling should be *valid* in a way that all points preserve their disjoint labels, but some labels may flip or all may resize to make room so that the obstacle can be inserted, and if needed, receives a similar label as other points.

Precisely, only two updating operations are allowed on the labels: (a) *flipping* ℓ_i over the edge containing its corresponding point p_i, and (b) *resizing* all labels to any length $\alpha \leqslant 1$, as long as any p_i remains on the mid-point of its edge. The flipped version of ℓ_i is denoted as $f(\ell_i)$ and the resized version of ℓ_i to length α is denoted as $r(\ell_i, \alpha)$. Moreover, all labels in the new updated labeling should have the same length with the biggest possible size. To construct the new labeling, we should also perform the fewest number of operations on the labels.

Avoiding a single obstacle in the r4PM and the 2PM models have first been introduced in [11,12]. Also, an incremental labeling for the 2PM model was presented in [13].

The existing brute-force solution for the above problems is based on the 2-SAT algorithm [3,9]. This method is independent of the existing valid labeling, and it basically solves the decision problem of whether there exists a valid labeling of size α for all points. The main idea of this solution is to assign variables x_i and $\bar{x_i}$ to two possible label locations for each point. If the obstacle falls into a label, then the corresponding literal is forced to be true or false. For each possible label conflict, we can write a clause of two literals. The whole labeling is then an instance of the 2-SAT problem which can be decided in $\mathrm{O}(n)$ time. Since there are at most $\mathrm{O}(n)$ possible label lengths for each instance of our problem, we can call the 2-SAT decision problem $\mathrm{O}(\lg n)$ times as a binary search on the possible values of α. This results in an $\mathrm{O}(n \lg n)$ time solution to generate the optimal labeling after each insertion of the obstacle. It is clear that this solution is unacceptable if the frequency of the obstacle appearance is quite large. For these problems, there exists no solution more efficient than the brute-force algorithm.

In this paper, we first deal with the POAR problem in Section 2. We define a data structure used in this problem and present the optimal algorithms for both 2PM and r4PM models. Overall, in the r4PM model, with a preprocessing of

[1] 'M' in these models comes from 'middle'.

$O(n^2)$ time and $O(n)$ space, our algorithm places an obstacle in $O(\lg n + k)$ time where k is the number of required flip operations (may be zero), or to decide on a new length for all labels to shrink to, needed to make room to insert the obstacle. We use specific properties of the 2PM model to reduce the preprocessing time to $O(n \lg n)$.

LOAS and POAS problems are handled in Section 3 where two new data structures are first introduced. Our algorithm for LOAS uses $O(n)$ space and $O(n \lg n)$ preprocessing time. The algorithm can then place and label a new obstacle in $O(\lg n + k)$ time, where k is the number of label flip operations to make free room for the new obstacle and its label. If the new obstacle cannot be labeled merely by flipping, we do overall labeling (discussed in Section 4) to find the optimal new labeling with shrunk labels in $O(n)$ time. The algorithm for POAS is actually a restricted version of that for LOAS which is explained in Section 3.4.

In Section 4, we propose two algorithms to optimally relabel all the points in $O(n)$ time and space using $O(n \lg n)$ preprocessing time.

2. POAR problem

Here we want to preprocess a given optimal labeling $\mathcal{L}$ of points $\mathcal{P}$, so that for any obstacle q, the new optimal labeling that avoids q is calculated efficiently. *Conflict graph* is a data structure used for preprocessing and is explained next. POAR algorithms for both 2PM and r4PM models are presented afterwards.

2.1. Data structure used: conflict graphs

A conflict graph $\mathcal{G} = (\mathcal{P} \cup \{p_0\}, D \cup B)$, is an edge-weighted directed multi-graph. The *domino* edge set D, representing all possible label flips, contains all edges (p_i, p_j) where $f(\ell_i)$ intersects with ℓ_j. There may be labels that have no intersections with other labels when flipped. To model these cases and simplify the definitions, we add an auxiliary point p_0 and include edges from all points to p_0 in D. The *blocking* edge set B contains the directed edges (p_i, p_j) where $f(\ell_i)$ intersects with $f(\ell_j)$. Intuitively, the blocking edges blocks the flippings. More formally,

$$D = \{(p_i, p_j) | f(\ell_i) \cap \ell_j \neq \emptyset\} \cup \{(p_i, p_0) | \; \forall p_i \in \mathcal{P}\}$$

and

$$B = \{(p_i, p_j) | f(\ell_i) \cap f(\ell_j) \neq \emptyset\}.$$

A sample labeled map is shown in Fig. 1(a) and its corresponding conflict graph is shown in Fig. 1(b).

The edge weight function $w(p_i, p_j)$ is based on the function $g(\ell_a, \ell_b)$ which returns the largest possible label length resolving, with resize operations only, any possible intersection between ℓ_a and ℓ_b. The value of $g(\ell_a, \ell_b)$ for any pair of non-intersecting labels is assumed to be one (unit length). For intersecting ℓ_a and ℓ_b, $g(\ell_a, \ell_b)$ is the maximum value of $\lambda \leqslant 1$ where $r(\ell_a, \lambda)$ is disjoint from $r(\ell_b, \lambda)$. The weight of a domino edge $(p_i, p_j) \in D$ is formally defined

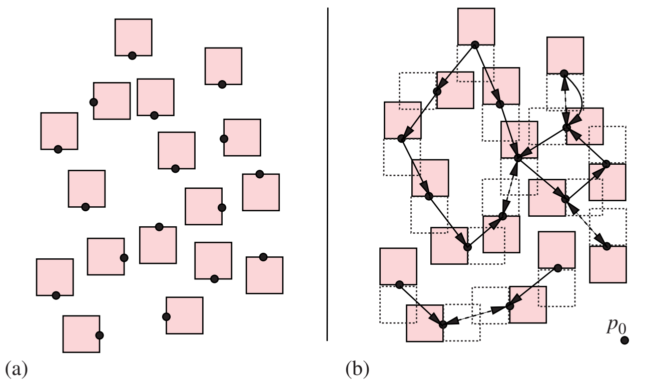

Fig. 1. (a) Initial labeled map. (b) The conflict graph: D edges are solid and B edges are dual-headed arrows. Edges ending at p_0 are not shown for simplicity.

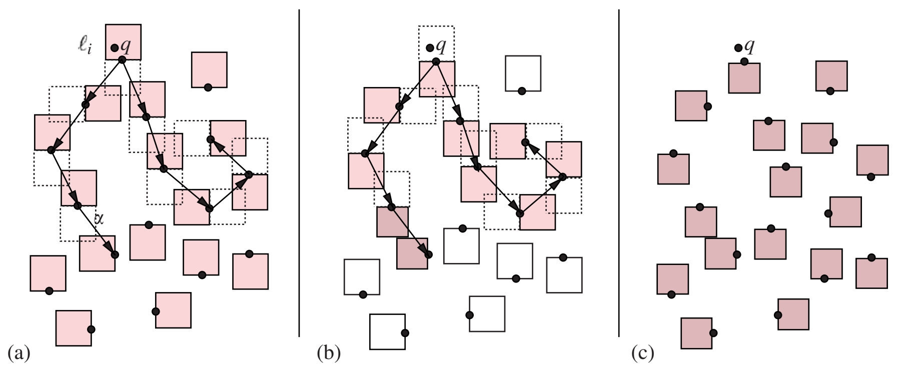

Fig. 2. For a given labeling and an obstacle q (inside ℓ_i), three cases are shown: (a) G_i^α and the domino edge bounding the maximum value of α; (b) the set of flipped labels and the pair of resized labels; and (c) the final optimal labeling with q. Note that p_0 and its edges are not drawn for simplicity.

as $w(p_i, p_j) = g(f(\ell_i), \ell_j)$ and that for a blocking edge $(p_i, p_j) \in B$ is defined as $w(p_i, p_j) = g(f(\ell_i), f(\ell_j))$. We also define the weight of all domino edges to p_0 as one.

Lemma 1. *Given a valid labeling of n points in the r4PM model, its conflict graph $\mathcal{G}$ can be built in* $O(n \lg n)$ *time and* $O(n)$ *space.*

Definition 2 (*Domino-reachable edge*). $(p_i, p_j) \in D$ is a *domino-reachable* edge from p_k in $\mathcal{G}$, if there is a directed simple path $\pi : p_k \rightsquigarrow p_i$ with all edges in D.

Definition 3 (α-*Terminating edge*). $(p_i, p_j) \in D$ is an α-*Terminating* edge from p_k, if (p_i, p_j) is a domino-reachable edge from p_k with a path π, where for each edge $e \in \pi$, $w(e) < \alpha$ and $w(p_i, p_j) \geqslant \alpha$. The path $p_k \rightsquigarrow p_j$ is also defined as an α-*terminating* path from p_k.

An α-terminating path π tells us that all labels on π can flip and the last one can resize to a size α without any intersection among these labels.

Definition 4 (α-*Critical*). $(p_i, p_j) \in D$ is an α-*critical* edge from p_k, if (p_i, p_j) is an α-terminating edge from p_k and $w(p_i, p_j) = \alpha$.

The graph G_i^α represents all labels that may be affected when ℓ_i is flipped and the label size is assumed to be α. We define $G_i^\alpha = (P_i^\alpha, D_i^\alpha)$ as a subgraph of $\mathcal{G}$ containing all α-terminating paths from p_i. That is, P_i^α and D_i^α are, respectively, the set of point vertices and domino edges on all α-terminating paths from p_i. It is obvious that G_i^α is unique. Also, G_i^α is a DAG in the 2PM model but may not be a DAG in r4PM. Fig. 2 depicts an initial labeling and an obstacle q that appears in ℓ_i for which G_i^α is also shown.

It is easy to see the following property.

Lemma 5. *If* $\alpha \leqslant \beta \leqslant 1$ *then* $G_i^\alpha \subseteq G_i^\beta$.

Definition 6 ($\mathcal{I}_i^\alpha$ *and* $\mathcal{B}_i^\alpha$). The set of internal nodes (not including the zero out-degree vertices) of G_i^α is denoted as $\mathcal{I}_i^\alpha$. The set of boundary edges (all edges that end in a zero out-degree vertex) of G_i^α is denoted as $\mathcal{B}_i^\alpha$.

As defined, the weights of all domino edges with end-points in $\mathcal{I}_i^\alpha$ are less than α and those for edges in $\mathcal{B}_i^\alpha$ are at least α. We are only concerned with *valid* G_i^α to be defined below.

Definition 7 (*Valid* G_i^α). G_i^α is said to be *valid* iff for all $p_i, p_j \in \mathcal{I}_i^\alpha$ and $(p_i, p_j) \in B$, we have $w(p_i, p_j) \geqslant \alpha$.

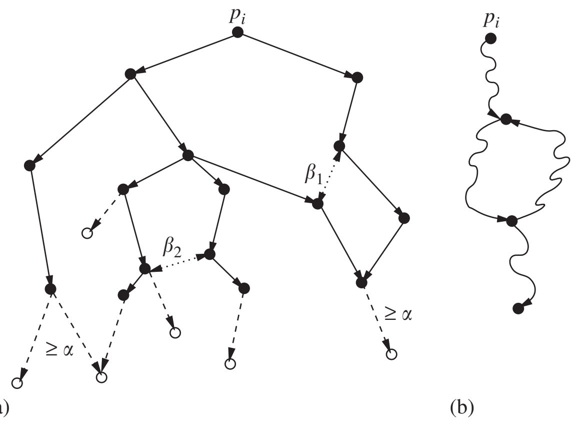

Fig. 3. (a) An instance of G_i^α for a labeling of 2PM model. Solid edges have weights $< \alpha$, dashed edges are $\mathcal{B}_i^\alpha$ with weights $\geqslant \alpha$, and dotted edges are blocking edges with different weights of β. Solid vertices are $\mathcal{I}_i^\alpha$. G_i^α is valid iff each value of β is $\geqslant \alpha$. D_i^α is the set of all solid and dashed edges. p_0 and all edges to this node are not shown. (b) G_i^α can be cyclic in r4PM model.

Fig. 3 depicts different properties of G_i^α graphs.
It is not hard to see the following lemma.

Lemma 8. *If there is no valid G_i^α, then there is no valid G_i^β for all $\beta > \alpha$.*

There are two possibilities when an obstacle intersects ℓ_i: (a) ℓ_i (and all other labels) can shrink to resolve the intersection (this case will be considered later) and (b) ℓ_i is flipped causing some other labels to flip or resize recursively.

Focusing on the latter case, to find an optimal labeling with ℓ_i flipped, we will see that we need to find a valid G_i^α with the maximum value of α. From any valid G_i^α we can construct a labeling of length α denoted by $\mathcal{L}_i^\alpha$, as follows:
(1) Flip label ℓ_j for each domino edge $(p_j, p_k) \in D_i^\alpha$.
(2) Resize ℓ_j and ℓ_k to length $w(p_j, p_k)$ for each boundary edge $(p_j, p_k) \in \mathcal{B}_i^\alpha$.
(3) Resize one or both labels ℓ_j and ℓ_k to length $w(p_j, p_k)$ for each blocking edge (p_j, p_k) with both ends in $\mathcal{I}_i^\alpha$.
(4) Resize all labels to the minimum label length of shrunk labels.
We will show that $\mathcal{L}_i^\alpha$ is an optimal labeling if and only if its corresponding G_i^α is valid and the value of its α is maximum.

2.2. POAR in r4PM model

The following is the main property of the optimal solution in the r4PM model.

Lemma 9. *$\mathcal{L}_i^\alpha$ is valid if and only if G_i^α is valid.*

The proof is given in the following lemmas:

Lemma 10. *A $\mathcal{L}_i^\alpha$ constructed from a valid G_i^α is also valid. Moreover, the label length in $\mathcal{L}_i^\alpha$ is α iff at least one of the following conditions holds*:
(1) *There exists an α-critical edge in G_i^α.*
(2) *There exists a blocking edge with both ends in $\mathcal{I}_i^\alpha$ and with the weight of α.*

Proof. The generated label lengths are, by the given construction method, at least α. These labels cannot have intersections with each other, since in such cases, we have already resized the overlapping labels to remove intersections. So, the produced labeling is valid. Any of the conditions mentioned in the lemma produces a label with the length of exactly α. The other side of the proof is obvious. □

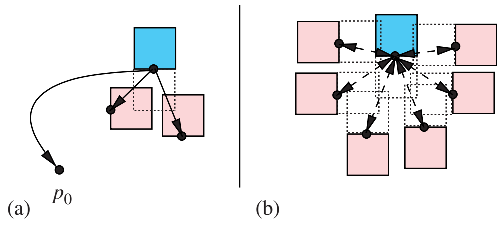

Fig. 4. The maximum number of edges from a given label (shown as the darker label) in (a) and (b), respectively.

Lemma 11. *For any valid labeling of $\mathcal{L}_i$ with label size of α and assuming that ℓ_i is flipped, there exists a corresponding valid G_i^α.*

Proof. We construct a valid G_i^α as follows. Enlarge all labels of $\mathcal{L}_i$ to at most unit length without generating intersecting labels. Define P^* as the set of all points whose labels have been flipped or resized, compared to the main $\mathcal{L}$ (including p_0). Let $E^* = \{(p_j, p_k) | p_j, p_k \in P^*, (p_j, p_k) \in D\}$ be the set of domino edges of $\mathcal{G}$ with both ends in P^*. Consider an α-terminating path from p_i in $\mathcal{G}$. This path is a subset of E^* since otherwise a label of length $< \alpha$ must exist in $\mathcal{L}_i$. So, the vertices and edges of G_i^α are subsets of P^* and E^*, respectively. Since $\mathcal{L}_i$ is valid, then there is no pair of intersecting flipped labels, or in other term, there is no blocking edge with both ends in P^* of length less than α. Hence, the G_i^α exists and is valid. □

From Theorem 9, we conclude what we claimed is true that flipping ℓ_i will generate an updated labeling of length α if and only if a valid G_i^α exists. Lemma 12 proves that we only need to check this for at most $O(n)$ values of α's, and this can be searched more effectively from the fact given in Lemma 8.

Lemma 12. *The optimal label length belongs to the set $W = \{w(p_j, p_k) | (p_j, p_k) \in D \cup B\}$, which has at most $O(n)$ elements.*

Proof. For the first part, assume that the optimal label length α is not in W. So, there should be a label, say ℓ_j, with length α in the optimal labeling. This label is resized to α to resolve an intersection with some other label, say ℓ_k. Obviously, the best resizing for these labels is $w(p_j, p_k) > \alpha$ and hence the value of α is not optimal, which is a contradiction.

For the second part, we count the number of edges in $\mathcal{G}$. It is obvious that for each $p_i \in \mathcal{P}$, there is at most three edges in D (Fig. 4(a)) and at most six edges in B (Fig. 4(b)) starting from p_i. Therefore, $|D \cup B| \leqslant 9n$ and hence the set W has $O(n)$ members at most. □

Now, assume that ℓ_i intersects with the obstacle, and G_i^α with the maximum value of α is constructed. Obviously, if the obstacle is outside of square $r(\ell_i, \alpha)$, then a single resize of ℓ_i to an appropriate value to resolve the intersection is the optimal labeling. Otherwise, if the obstacle is inside $r(\ell_i, \alpha)$, it should not intersect any of the flipped labels (which are at most two) generated from G_i^α. We find and store these intersecting flipped labels in the preprocessing phase and store it along with the maximum value of α at each node p_i. Equally, the locus of the obstacles that force ℓ_i to flip, which is an orthoconvex polygon and is denoted by *flipping region*, can be computed and stored in the processing phase. Using this data at p_0, the algorithm is able to decide between flipping or resizing ℓ_i in $O(1)$ time.

2.2.1. The optimal algorithm

The key routine in the optimal algorithm is CONSTRUCT(G_i^α) that generates G_i^α with the maximum value of α for any given label ℓ_i. This routine has two purposes: in the preprocessing phase, it is used to find the flipping region and in the query part of the algorithm, we extract the required flip and resize operations from the calculated subgraph. We will show that this routine finds G_i^α in $O(k)$ amortized time where $k = |G_i^\alpha|$.

The preprocessing phase: This phase consists of the following steps.

(1) Build the conflict graph $\mathcal{G}$.
(2) Create a sorted list of all edge weights $\alpha_i = w(e)$, where $\alpha_1 \leqslant \alpha_2 \leqslant \cdots < 1$.

(3) Construct a point location data structure for the initial labels [15].
(4) For each ℓ_i, construct G_i^α with the maximum value of α, and calculate and store the flipping region of ℓ_i.

Lemma 13. *The preprocessing phase needs* $O(n^2)$ *time and* $O(n)$ *space.*

Proof. The conflict graph can be constructed using a simple vertical sweep line algorithm keeping track of all intersecting labels with the sweep line in $O(n \lg n)$ time. The value of $w(e)$ can also be computed in $O(1)$ time for each edge. Steps 2 and 3 also take $O(n \lg n)$ time. To calculate the flipping region in Step 4, we have to build G_i^α that needs $O(n)$ time for each of n label. So, this step takes $O(n^2)$ time total. Thus, the overall time required in the preprocessing phase is $O(n^2)$.

The conflict graph has $O(n)$ vertices and edges and there are $O(n)$ different values for α (Lemma 12), hence, the first three steps require $O(n)$ space. The point location data structure also uses $O(n)$ space and the cost of saving the flipping region for any given point is $O(1)$ space. So, the overall required space is $O(n)$. □

CONSTRUCT(G_i^α) *routine*: Given a label ℓ_i, this routine starts with building the subgraph $G_i^{\alpha_1}$. Then, at each iteration, the routine builds $G_i^{\alpha_{j+1}}$ by adding some vertices and edges (possibly empty) to $G_i^{\alpha_j}$. The following properties are used in constructing $G_i^{\alpha_{j+1}}$.

(1) $G_i^{\alpha_1} \subseteq G_i^{\alpha_2} \subseteq G_i^{\alpha_3} \subseteq \cdots$ (definition of $G_i^{\alpha_j}$),
(2) an edges in $G_i^{\alpha_{j+1}} - G_i^{\alpha_j}$ which is connected to a leaf vertex of $G_i^{\alpha_j}$, has a weight of at least α_j (due to the definition of α_j-terminating path from p_i in Definition 3),
(3) the value $\beta_{j+1} = \min(\{w(u,v) | (u,v) \in \mathcal{I}_i^{\alpha_{j+1}}\})$ is an upper bound of the optimal label length (due to Definition 7, if $\beta_{j+1} < \alpha_{j+1}$ the subgraph is not valid),
(4) $G_i^{\alpha_{j+1}}$ is valid if and only if $\beta_{j+1} \geqslant \alpha_{j+1}$ (Definition 7), and
(5) by considering $G_i^{\alpha_{j+1}} - G_i^{\alpha_j}$, the recursive definition of $G_i^{\alpha_{j+1}}$ (definition of $G_i^{\alpha_{j+1}}$ and Definition 7) is as follows:

$$G_i^{\alpha_{j+1}} = G_i^{\alpha_j} \cup \{G_k^{\alpha_{j+1}} | (p_l, p_k) \in \mathcal{B}_i^{\alpha_j}, w(p_l, p_k) = \alpha_j\}.$$

Using the above recursive definition of $G_i^{\alpha_{j+1}}$, the following simple incremental algorithm is proposed.

```
                    CONSTRUCT($G_i^\alpha$)
Input: A conflict graph $\mathcal{G}$, and a label $\ell_i$,
Output: The subgraph $G_i^\alpha$ with the maximum value of $\alpha$
Algorithm:
    Let $j = 1$, build $G_i^{\alpha_1}$.
    while true do
    (a) Compute $\{G_k^{\alpha_{j+1}} | (p_l, p_k) \in \mathcal{B}_i^{\alpha_j}, w(p_l, p_k) = \alpha_j\}$ and build $G_i^{\alpha_{j+1}}$.
    (b) Compute sets $\mathcal{B}_i^{\alpha_{j+1}}$, $\mathcal{I}_i^{\alpha_{j+1}}$, and $\beta_{j+1}$.
    (c) if $\beta_{j+1} < \alpha_{j+1}$ then
            return $G_i^{\alpha_j}$ and terminate.
    (d) Let $j = j + 1$.
    end while
```

Lemma 14. $G_i^{\alpha_j}$ *for all values of* α_j *can be constructed in the ascending order of* α_j *in overall* $O(n)$ *time.*

Proof. Any edge in the conflict graph is visited at most a constant number of times: a domino edge is visited at most once, and a blocking edge is visited at most twice (a blocking edge is checked whenever one of its ends is added to the internal vertex set). With an appropriate data structure for maintaining the sets $\mathcal{I}$ and $\mathcal{B}$, each visit to an edge can be implemented in $O(1)$ time. □

Main body of the algorithm: Using the CONSTRUCT(G_i^α) routine, the label updating algorithm is as follows.

The Optimal POAR Algorithm in r4PM Model

For an obstacle $q = (x, y)$ do the following steps:

(1) Locate the label ℓ_i containing q.
(2) **if** no such label exists **then** $\mathcal{L}$ is the optimal labeling and **terminate**.
(3) **if** q is not in the flipping region of ℓ_i, **then** resize ℓ_i to obtain the optimal labeling.
(4) Call CONSTRUCT(G_i^α) and write the required operations to construct $\mathcal{L}_i^\alpha$.

The above algorithm along with Lemma 14 yields the following theorem:

Theorem 15. *An optimal updated r4PM labeling to avoid a single obstacle can be constructed in* $O(\lg n+k)$ *time, where k is the minimum number of update operations. The algorithm uses* $O(n^2)$ *time and* $O(n)$ *space for the preprocessing.*

2.3. POAR in 2PM model

Obviously, 2PM is a restricted version of the r4PM model and the previous algorithm can also be used in this case. However, using some specific properties of the conflict graph in the 2PM labeling, we will show that the preprocessing phase takes $O(n \lg n)$ time and $O(n)$ space, instead.

2.3.1. Properties of the conflict graph 2PM model

The simple but important property of the 2PM conflict graph is that its edges are all in “upwards” or “downwards” directions, assuming that the vertices of the graph are located on a plane each on the coordinates of its point. We denote the upwards edges by $E^\uparrow$ and the downwards edges by $E^\downarrow$. The following lemmas immediately hold.

Lemma 16. *All domino-reachable edges from any point $p_i \in \mathcal{P}$ belong to either $E^\uparrow$ or $E^\downarrow$.*

Lemma 17. *The edges of any G_i^α either belong to $E^\uparrow$ or $E^\downarrow$.*

Lemma 17 also shows that the edges D of the conflict graph are acyclic. So, when an ℓ_i-intersecting obstacle is inserted, no flipped label generated by the respective G_i^α will intersect the obstacle. Therefore, the flipping region can be discarded and the decision between cases of resizing or flipping ℓ_i is simplified. The maximum value of α where p_i has a valid G_i^α can be stored in the conflict graph as the vertex weight of p_i, denoted by $w(p_i)$. Besides, we can calculate the vertex weight of all vertices in $\mathcal{G}$ at each vertex without explicitly building the related G_i^α. We define the weight of p_0, the only vertex with no outgoing D edge, as one.

For vertices with some outgoing D edges, consider an edge $(p_i, p_j) \in E^\uparrow$ and assume that ℓ_i is flipped. There are two alternatives to remove the intersection between $f(\ell_i)$ and ℓ_j: (1) resize both intersecting labels to some smaller length or (2) flip ℓ_j and resolve the newly introduced intersection recursively (if applicable). In the former case, the minimum generated label length is $w(p_i, p_j)$ (according to the edge weight definition), and in the later case is $w(v_j)$, recursively. We summarize the above description in a function, denoted by h, as follows:

$$h(v_i, v_j) = \begin{cases} \max(w(p_i, p_j), w(p_j)), & (p_i, p_j) \in D, \\ 1, & (p_i, p_j) \in B. \end{cases}$$

Using above function, the vertex weights can precisely be defined as

$$w(p_i) = \begin{cases} 1, & i = 0, \\ \min\{h(p_i, p_j) | (p_i, p_j) \in D \cup B\}, & i > 0. \end{cases}$$

Using the above recursive definition of the vertex weight, it is easy to verify that a simple bottom-up algorithm is able to calculate all vertex weights in $O(n)$ time. This bottom-up procedure reduces the $O(n^2)$ time used by the Step 2.2.1 of the preprocessing phase in the r4PM model to $O(n)$ time, hence the overall preprocessing time is reduced to $O(n \lg n)$ time.

The updating algorithm in the 2PM model uses the same CONSTRUCT(G_i^α) subroutine as in the r4PM model and is given below:

The Optimal POAR Algorithm in 2PM

For a given obstacle $q = (x, y)$ do the following steps:

```
(1) Locate the label $\ell_i$ containing $q$.
(2) if no such label exists then $\mathcal{L}$ is the optimal labeling and terminate.
(3) if $r(\ell_i, w(p_i))$ does not intersect $q$, then resize $\ell_i$ to resolve its intersection with $q$ and terminate.
(4) Call CONSTRUCT($G_i^\alpha$) and write the required operations according to $G_i^\alpha$.
```

Theorem 18. *An optimal updated 2PM labeling to avoid any single obstacle can be constructed in* $O(\lg n + k)$ *time, where k is the minimum number of update operations using* $O(n \lg n)$ *preprocessing time and* $O(n)$ *space.*

3. LOAS and POAS problems

In this section, we present and solve the LOAS and POAS problems. The algorithms provided to solve these problems uses a relabeling algorithm whenever a relabeling of all points is required. Two relabeling algorithms are presented in Section 4.

For these algorithms, either enough room for insertion of the obstacle is created by some label flips or all points are relabeled in the presence of the obstacle. That is why we only need the unweighted version of the conflict graphs which we call *domino graphs* and denoted by $\mathcal{DG}$. To update the domino graph we need another data structure called *adjacency graphs* to be defined below.

3.1. Data structures used: adjacency graphs

The adjacency graphs are used to both update the domino graphs, and to relabel all points whenever required.

The distance between two points $p_i = (x_i, y_i)$ and $p_j = (x_j, y_j)$ is defined as $\Delta(p_i, p_j) = \max(|x_i - x_j|, |y_i - y_j|)$ (i.e., the distance in L_∞ metric).

Definition 19 (*Local neighbors*). Assuming a valid labeling of size α over $\mathcal{P}$ exits, the *local neighbors* of a point p_i is defined as the set of points with the distance less than 2α from p_i.

Lemma 20. *There are at most* 11 *local neighbors for any point in* $\mathcal{P}$.

Proof. In a valid labeling of length α, label candidates of a point p, form a α by 2α rectangle. Obviously, a label intersecting any of these candidates, must reside completely inside a 3α by 4α rectangle around the α by 2α rectangle. The area of the 3α by 4α rectangle is $12\alpha^2$ and has room for 12 labels. Hence at most 11 local neighbors may exist for any p. □

An adjacency graph $\mathcal{H} = (\mathcal{P}, \mathcal{E})$ is a weighted directed graph with $\mathcal{P}$ as its vertices. A directed edge (p_i, p_j) belongs to $\mathcal{E}$ if and only if p_j is a local neighbor of p_i. The weight of (p_i, p_j), denoted by $w(p_i, p_j)$, is defined as $\Delta(p_i, p_j)$ (i.e., the distance between p_i and p_j).

Lemma 21. *Given a set of n points*, $\mathcal{H}$ *can be built in* $O(n \lg n)$ *time and* $O(n)$ *space.*

To update $\mathcal{H}$ when a new point p is introduced, we construct and use a nearest neighborhood data structure on all points using L_∞ metric. With this, all local neighbors of p can be found in $O(\lg n)$ time. But, inserting p may change the local neighbor set of its neighbors which can be updated in $O(1)$ time since p has $O(1)$ local neighbors at most.

Lemma 22. *Given an adjacency graph* $\mathcal{H}$, *a new point can be inserted into* $\mathcal{H}$ *in* $O(\lg n)$ *time. Also, the edges of* $\mathcal{H}$ *can be updated in the same time bound.*

The adjacency graph is used to update the domino graph after a set of labels is flipped. We know that $\mathcal{DG}$ is a subgraph of $\mathcal{H}$, so the update procedure is as follows. First, all edges attached to the points with flipped labels in $\mathcal{DG}$ are deleted. Then, all edges attached to these points in $\mathcal{H}$ are checked and inserted into D or B of $\mathcal{DG}$, if the edge definitions are satisfied. Since there are $O(k)$ edges in $\mathcal{H}$ for a set of k flipped labels (according to Lemma 20), this update takes $O(k)$ time. Therefore,

Lemma 23. *Updating the domino graph $\mathcal{DG}$ after k label flips requires $O(k)$ time, using the adjacency graph $\mathcal{H}$.*

3.2. The main ideas of the algorithms

We know that if $(p_i, p_j) \in D$, then flipping ℓ_i may cause flipping a set of labels whose points are reachable from p_i via domino edges. Here, we denote the domino reachable points from p_i as P_i and the induced subgraph of D edges on P_i as G_i. Note that G_i is the same as G_i^1 in the conflict graph of $\mathcal{G}$ if the optimal label length is 1. Obviously, ℓ_i can flip if and only if there is no blocking edge with both ends in P_i. Using a simple BFS-like algorithm, we can easily see that, for any point p_i, G_i can be constructed in $O(|G_i|)$ time.

Given a point p, and its label direction $d(p)$ in LOAS, there are two label candidates for p, denoted by ℓ_p^a and ℓ_p^b. These are a pair of top/bottom candidates if $d(p)$ is horizontal, or a pair of left/right candidates if it is vertical. Since all labels are squares with the same size, using a point location data structure [15], all intersecting labels with ℓ_p^a and ℓ_p^b can be found in $O(\lg n)$ time. To label p with ℓ_p^a without shrinking any labels, all labels intersecting ℓ_p^a should be flipped. So, we define G_p^a (resp. G_p^b) as the union of all G_i's where ℓ_i intersects with ℓ_p^a (resp. ℓ_p^b). Moreover, similar to the conflict graph, G_p^a is said to be valid if: (a) there is no blocking edge with both ends in G_p^a (i.e., no pairs of intersecting flipped labels) and (b) ℓ_p^a has no intersection with the flipped version of the labels in G_p^a. It is then easy to see the following lemma.

Lemma 24. *A new point p can be labeled by ℓ_p^a iff G_p^a is valid. Moreover, G_p^a can be found and validated in* $O(|G_p^a|)$ *time.*

Both G_p^a and G_p^b should be constructed and checked for validity. To generate the minimum number label flips, the smaller subgraph (the one with the smaller number of vertices) should be chosen for generating the final label flip sequence. But, since the sizes of G_p^a and G_p^b are not known in advance, both should be processed simultaneously such that the first validated candidate terminates the decision process.

Lemma 25. *Given a new point p, $d(p)$, and $\mathcal{DG}$ of the current labeling, p can be labeled in* $O(\lg n + k)$ *time with the same label length, if possible, where k is the minimum number of label flip operations. Moreover, $\mathcal{DG}$ is updated in* $O(k)$ *time for further point insertions.*

3.3. LOAS: incremental labeling

By summarizing the lemmas given in the previous sections, LOAS optimal updating algorithm can be constructed in two phases: a preprocessing phase and a query phase. The following data structures should be built in the first phase:
(1) The domino graph $\mathcal{H}$.
(2) The adjacency graph $\mathcal{DG}$.
(3) A nearest neighborhood data structure on all points.

For each new point p arrival with its label direction $d(p)$, the following steps are taken in the query phase:
(1) Insert p into the nearest neighborhood data structure.
(2) Update $\mathcal{H}$.
(3) Generate and validate G_p^a and G_p^b in parallel.
(4) **If** both G_p^a and G_p^b fail, **then** relabel all points, **otherwise**, generate the flipping sequence.
(5) Update $\mathcal{DG}$.

As shown in the previous sections, the preprocessing phase takes $O(n \lg n)$ time and $O(n)$ space. Also, inserting and labeling each new point takes $O(\lg n + k)$ time, if there exists a series of k label flips to label the new point, and $O(n)$ time (to be explained in the next section) if a relabeling is required.

3.4. POAS in r4PM model

The solution to POAS in r4PM comes from the solution to LOAS with minor modifications. The basic idea here is to consider a point obstacle as a new point with a zero-length label. There are also regular non-obstacle points as before that take labels with optimal length. Since all properties of the LOAS are also satisfied in POAS, one can modify this algorithm and develop a solution to POAS in the r4PM labeling. Obviously, the response time of the POAS algorithm would be the same that for LOAS. In the rest of this section, we only consider the required modifications to LOAS as follows.

In POAS, each point in $\mathcal{P}$ carries a special flag to show if it is a point obstacle (i.e., it should be labeled by a zero-length label) or a labeled obstacle.

There is no special note in building of the domino graph or adjacency graph, but we consider some useful properties as follows:

(1) In the domino graph, if an obstacle point p_j, is inside ℓ_i, we assume that ℓ intersects with both ℓ_j and $f(\ell_j)$.

(2) There is no edge between two obstacle points in the domino graph.

A consideration should be taken in the construction of the adjacency graph in POAS (which is used in relabeling). Local neighbors of a point p, consists of two sets of points: (a) the 11 nearest labeled obstacles and (b) the nearest point obstacle in each of half planes above, right, below and left of p. This way, we will have at most 15 local neighbors for each point, and the relabeling results are hold for POAS.

Theorem 26. *Given a set of points $\mathcal{P}$, and its optimal labeling $\mathcal{L}$ in the r4PM model, using an $O(n)$ space and $O(n \lg n)$ preprocessing time, an obstacle points can be inserted into $\mathcal{P}$ in $O(\lg n + k)$ time, when k label flips makes this possible. Otherwise, an $O(n)$ time algorithm is needed to optimally relabel the map with a smaller label length.*

4. Overall relabeling

In this section, we present two algorithms to relabel all points in $O(n)$ time. Both algorithms require $O(n \lg n)$ preprocessing time and $O(n)$ space. The first algorithm, named *incremental relabeling*, needs a valid labeling of all points except the arriving one, say p. Incremental algorithm modifies the valid labeling by flipping some labels and resizing all labels so that at least one label candidate of p can be placed without any label intersection. This way, most of the labels remain on their old places and just shrink in size.

The second algorithm, the *total relabeling*, assigns labels from scratch to all points and do not rely on a previous labeling. So, a significant difference is expected for the outcome of these two algorithms, which is the price for the generality of the second algorithm.

4.1. Incremental relabeling

Given a set of labeled points, assume that a new point p, arrives and there is no sequence of label flips making free space to label p. p has two label candidates, say ℓ_p^a and ℓ_p^b. The optimal label length in each case, as defined in POAR in r4PM, is equal to the vertex weight of p. But, the problem is that the domino graphs are unweighted. G_p^a and G_p^b can be constructed but, due to the definition given in 3.2, are both invalid. But if edge and vertex weight functions are used temporarily, the optimal value of the label length can be computed as follows.

Assume that ℓ_p^a is fixed on the labeling, and is only allowed to shrink. We insert p and its corresponding edges to the domino graph. Now, we construct G_p^a and use the bottom up recursive definition, given in Section. 2.1, to compute the vertex weight of p. Obviously, this weight equals to the optimal label length, if ℓ_p^a is fixed, and can be computed in $O(|G_p^a|)$ time. Using the same routine, the optimal label length can also be computed for labeling p by ℓ_p^b by spending another $O(|G_p^b|)$ time. Obviously, the label candidate generating the greater label length, say α, is the optimal choice to label p. Using the edge weight function, we can use the CONSTRUCT(G_i^α) idea for p and compute the set of flipping

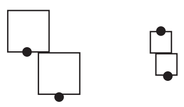

Fig. 5. Touching cases in the 2PM model.

labels. It is obvious that by flipping these labels and shrinking labels to length α, p can be safely labeled by at least one of its candidates.

Theorem 27. *Given a $\mathcal{P}$, its optimal labeling $\mathcal{L}$, and its domino and adjacency graph, an optimal labeling of $\mathcal{P}$ and a new given point p, can be constructed in $\mathrm{O}(n)$ time and space, with $\mathrm{O}(n \lg n)$ preprocessing time. Moreover, the domino graph of $\mathcal{P} \cup \{p\}$ can be updated in $\mathrm{O}(n)$ time.*

There is only one implementation issue here: the domino graph needs a refresh after a label shrink, since some previously intersecting labels may not be intersecting anymore. Removing these edges is an $\mathrm{O}(n)$ time step after a label shrink. But, to speed up the shrink process, this time consuming step can be delayed to a later time, as explained in the next paragraph.

We define an edge as *valid* if it meets its definition criteria. Upon visiting an edge, we check its validity and remove each invalid edge from the graph. In this scheme, there may exist more than $\mathrm{O}(n)$ invalid edges in the domino graph. Since an invalid edge will never be validated, it can be removed by spending the $\mathrm{O}(1)$ amortized cost saved when we do not remove it after a label shrink. So, Theorem 27 still holds but with the amortized times.

4.2. *Total relabeling*

Given an optimal labeling, labels can be categorized into two sets of *forced* and *free* labels based on the following definition. Assume that γ_i is the location of ℓ_i in an optimal labeling of $\mathcal{P}$. We define ℓ_i as a forced label if in every optimal labeling, ℓ_i is located at γ_i, and it is said to be free, otherwise. In some labeling models (i.e., like 1P, 2P, 2PM and r4PM), if the optimal label length is known in advance, the set of forced labels can be found easily. For example, in a 2PM labeling of length α, the labels of every pair of points with distance less than α are forced. Consider a point p, which one of its label candidates has an intersection with a forced label. Clearly, the other label candidate of p is also forced. So, the set of forced labels can simply be calculated using a recursive algorithm.

The main idea of the total relabeling is as follows. First, construct a set of optimal label length candidates. Start with the minimum candidate and construct the set of forced labels. Then, grow the label length, according to the candidates list, and expand the set of forced labels until an unavoidable label intersection between forced labels occurs. Decrease label length to remove the intersection and take this length as the optimal label length. Finally, relabel the unlabeled points, which are not very close to each other, with a greedy algorithm. In this section, we consider the total relabeling in the 2PM model in detail but, this method, with minor changes, can be applied to the 1P, 2P and r4PM models.

Consider an edge (p_i, p_j) in the edges of the adjacency graph with weight of δ (i.e., $\Delta(p_i, p_j) = \delta$). Labels of p_i and p_j may touch each other in a valid optimal labeling only in the following two cases, referred to as *touching cases*:

(a) at label length δ, when both labels are attached on above or below of their related points and
(b) at label length $\delta/2$, where the lower point is attached to its label from bottom, and the other point is attached to the top of its label.

Fig. 5 depicts these two cases.

In any optimal labeling, there is at least one pair of touching labels. So, this observation gives two possible candidates for the optimal label length for each edge of $\mathcal{E}$. For an edge $(p_i, p_j) \in \mathcal{E}$, denote these two candidates as $\delta^1_{i,j}$ and $\delta^2_{i,j}$ where $\delta^1_{i,j} < \delta^2_{i,j}$. Let Γ be a sorted sequence of all possible label length candidates in ascending order.

The second case of the touching labels, as defined above, gives another clue: for any labeling with length greater than $\Delta(p_i, p_j)/2$, both p_i and p_j have only one valid way to be labeled, hence their labels are forced.

Moreover, if a forced label ℓ_i of length α intersects a label candidate of an unlabeled point, say q, then q looses the intersecting candidate and hence its other label candidate will be forced. Obviously, this new forced label may force other labels of its local neighbors, recursively. Assuming that the label length is α, let $L(\ell_i, \alpha)$ denote the set of all forced labels after forcing ℓ_i, and $P(\ell_i, \alpha)$ denote all points with forced labels in $L(\ell_i, \alpha)$. $L(\ell_i, \alpha)$ is called valid if it has no pair of intersecting labels and also causes no intersection with other previously forced labels.

A forced label ℓ_i of length α may have enough room to expand. Assume that $\beta > \alpha$ be the smallest value in Γ such that $L(\ell_i, \alpha) \neq L(\ell_i, \beta)$. If $L(\ell_i, \beta)$ is valid, then the following lemmas hold.

Lemma 28. *$L(\ell_i, \alpha) \subset L(\ell_i, \beta)$, if $\alpha < \beta$.*

Lemma 29. *There exists a directed edge $(p_j, p_k) \in \mathcal{E}$ where $p_j \in P(\ell_i, \alpha)$ and $p_k \notin P(\ell_i, \alpha)$. Moreover, one of the $\delta^1_{j,k}$ or $\delta^2_{j,k}$ is equal to β.*

Proof. Obviously, if no such edge exists, $L(\ell_i, \alpha) = L(\ell_i, \beta)$ which contradicts the assumption. Also, the edge (p_j, p_k) does not force ℓ_k at length α, but forces ℓ_k at length β. So, one of the touching cases should occur at this edge, hence one of its optimal label length candidates should be equal to β. □

Using the above lemma, $L(\ell_i, \beta)$ can be built by checking all outgoing edges of $P(\ell_i, \alpha)$ with one of its δ^1 or δ^2 values equals to β, and inserting their forced label sets into $L(\ell_i, \alpha)$.

Let $\Gamma = \langle d_1, d_2, \ldots \rangle$. Also, assume that $\eta(d_i)$ is the set of all edges generating d_i. Obviously, for the label length of d_1, there is no forced label and all points may receive either of their label candidates. In step i, it is assumed that the label length is incremented by a small value ε, where $d_i + \varepsilon < d_{i+1}$. A labeling of length $d_i + \varepsilon$ exists for all points if any directed edge (p_j, p_k) in $\eta(d_i)$ can be verified as follows, meaning that p_j and p_k can be labeled with sizes of $d_i + \varepsilon$ with no problem.

(a) *Check*: If both ℓ_j and ℓ_k are forced and they have intersection with each other for length $d_i + \varepsilon$ (i.e., $\delta^2_{j,k} = d_i$), the verification fails.
(b) *Expand*: If ℓ_j is forced and intersects with a label candidate of p_k, then the other label candidate of p_k is forced. Formally, $L(\ell_k, \mathrm{d}_1 + \varepsilon)$ should be calculated and inserted into the set of forced labels. If the new forced labels have intersection with other forced labels, the verification fails.
(c) *New*: If neither p_j nor p_k has a forced label but form a touching case at label length d_i, then build the forced sets for both p_j and p_k. If the new forced labels has any intersection, the verification fails.
(d) *Success*: In all other cases, the verification of the edge (p_j, p_k) succeeds.

If all edges in $\eta(d_i)$ are successfully verified, then a labeling of length $d_i + \varepsilon$ exists and it is easy to show that a labeling of length d_{i+1} also exists.

When above algorithm finds an intersection in the ith iteration, then there exists a set of forced labels of length d_i and a set of unlabeled points. Since all unlabeled points have distances of at least d_i from their neighbors, they can be labeled with a simple heuristic: place a label of length d_i above all unlabeled points.

Lemma 30. *Given an adjacency graph of a set of n points $\mathcal{P}$, the optimal labeling of $\mathcal{P}$ can be calculated in* $\mathrm{O}(n)$ *time and* $\mathrm{O}(n)$ *space.*

Proof. Each edge has two candidates in Γ and may be verified at most two times in the algorithm. Moreover, each edge may be considered once more whenever one of its end vertices is processed. Since each point has at most 11 local neighbors, then each edge is considered at most $\mathrm{O}(1)$ times. Therefore, the overall time is $\mathrm{O}(n)$. □

After building a new labeling for $\mathcal{P}$, the domino graph should also be updated.

Lemma 31. *Having $\mathcal{P}$, the optimal labeling and the adjacency graph, the updated domino graph $\mathcal{DG}$, can be constructed in* $\mathrm{O}(n)$ *time.*

Proof. To build $\mathcal{DG}$, it is sufficient to check all labels and see if the flipped version of that label may satisfy any edge condition of $\mathcal{DG}$. Using the adjacency graph, possible intersections between any flipped label and its local neighbors can be found in $\mathrm{O}(1)$ time, hence all possible intersections for the flipped version of n labels can be tested in $\mathrm{O}(n)$ time. □

Since the sorted list of the optimal label length candidates can be updated in $O(\lg n)$ using a balanced tree, the following theorem can be concluded.

Theorem 32. *Given a $\mathcal{P}$ and its adjacency graph, an optimal labeling of $\mathcal{P}$ can be constructed in $O(n)$ time, with $O(n \lg n)$ preprocessing time and $O(n)$ space. Moreover, the domino graph of $\mathcal{P}$ can be constructed in $O(n)$ time.*

5. Conclusion

In this paper, we introduced the problem of label updating when a series of point-shaped obstacles are introduced on random locations in a given map. We considered this problem in the 2PM and r4PM labeling models in three settings: (1) (POAR) each obstacle is removed afterwards, (2) (POAS) obstacles remain on the map, and (3) (LOAS) obstacles receive similar labels and remain on the map. We assumed that labels are only permitted to flip or resize to avoid the obstacle(s) while maximizing the label length. In the POAR setting, we proposed an $O(n^2)$ preprocessing time with an $O(n)$ space preprocessing algorithm for the r4PM model to place an obstacle in $O(\lg n + k)$ time, where k is the minimum number of label updates in the map. Then, we showed that using the specific properties of the 2PM model, the processing time can be reduced to $O(n \lg n)$ time.

The solution of the POAS is based on the solution of the LOAS by assuming that the obstacles are special points with zero-length labels. Both LOAS and POAS solutions use $O(n)$ space and $O(n \lg n)$ preprocessing time. A new point (i.e., an obstacle), in both settings, can be inserted into the map in $O(\lg n + k)$ time if k label flips can make enough room to place and/or label the new point. Otherwise, we suggested two $O(n)$ time and space relabeling algorithms with $O(n \lg n)$ preprocessing time that generate a new optimal labeling with smaller labels for all current points in the map.

This problem can also be considered when a point is removed from the map that may cause label expansions. Also, it is challenging to consider both point arrival and removals and come up with $o(n \lg n)$ algorithms for the general case.

References

[1] S. Doddi, M.V. Marathe, A. Mirzaian, B.M. Moret, B. Zhu, Map labeling and its generalizations, in: Proc. Eighth ACM-SIAM Symp. on Discrete Algorithms (SODA'97), 1997, pp. 148–157.

[2] R. Duncan, J. Qian, A. Vigneron, B. Zhu, Polynomial time algorithms for three-label point labeling, Theoret. Comput. Sci. 296 (1) (2003) 75–87.

[3] M. Formann, F. Wagner, A packing problem with applications to lettering of maps, in: Proc. Seventh Annu. ACM Symp. on Computational Geometry (SoCG'91), 1991, pp. 281–288.

[4] C. Iturriaga, Map labeling problems, Ph.D. Thesis, University of Waterloo, 1999.

[5] G.W. Klau, P. Mutzel, Optimal labelling of point features in the slider model, in: D.-Z. Du, P. Eades, V. Estivill-Castro, X. Lin, A. Sharma (Eds.), Proc. Sixth Annual Internat. Computing and Combinatorics Conference (COCOON'00), Lecture Notes in Computer Science, Vol. 1858, Springer, Berlin, 2000, pp. 340–350.

[6] G.W. Klau, P. Mutzel, Optimal labeling of point features in rectangular labeling models, Math. Programming (Ser. B) (2003) 435–458.

[7] M. van Kreveld, T. Strijk, A. Wolff, Point labeling with sliding labels, Comput. Geometry: Theory Appl. 13 (1999) 21–47.

[8] J. Marks, S. Shieber, The computational complexity of cartographic label placement, Technical Report TR-05-91, Harvard CS, 1991.

[9] S.-H. Poon, C.-S. Shin, T. Strijk, A. Wolff, Labeling points with weights, in: P. Eades, T. Takaoka (Eds.), Proc. 12th Annu. Internat. Symp. on Algorithms and Computation (ISAAC'01), Lecture Notes in Computer Science, Vol. 2223, Springer, Berlin, 2001, pp. 610–622.

[10] C.K. Poon, B. Zhu, F. Chin, A polynomial time solution for labeling a rectilinear map, Inform. Process. Lett. 65 (4) (1998) 201–207.

[11] F. Rostamabadi, M. Ghodsi, A fast algorithm for updating a labeling to avoid a moving point, in: Proc. 16th Canad. Conf. on Computational Geometry (CCCG'04), 2004, pp. 204–208.

[12] 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.

[13] F. Rostamabadi, M. Ghodsi, Incremental labeling in 2PM model, in: Computer Society of Iran Computer Conference 2006 (CSICC'06), 2006, pp. 9–13.

[14] S. Roy, P.P. Goswami, S. Das, S.C. Nandy, Optimal algorithm for a special point-labeling problem, in: M. Penttonen, E.M. Schmidt (Eds.), Proc. Eighth Scandinavian Workshop on Algorithm Theory (SWAT'02), Lecture Notes in Computer Science, Vol. 2368, Springer, Berlin, 2002, pp. 110–120.

[15] N. Sarnak, R.E. Tarjan, Planar point location using persistent search trees, Comm. ACM 29 (7) (1986) 669–679.

[16] T. Strijk, M. van Kreveld, Labeling a rectilinear map more efficiently, Inform. Process. Lett. 69 (1) (1999) 25–30.

[17] T. Strijk, M. van Kreveld, Practical extensions of point labeling in the slider model, GeoInformatica 6 (2) (2002) 181–197.

[18] T. Strijk, A. Wolff, Labeling points with circles, Internat. J. Comput. Geometry Appl. 11 (2) (2001) 181–195.