# 计算几何实验报告

## Point Avoiding

张力(2004211033)
孙洛(2004310411)
徐俊(2004310436)
{tension00, sunluo00, xujun00}@mails.tsinghua.edu.cn
计算机科学与技术系 清华大学 北京 100084

# 目录

# 图表目录

# 1 背景介绍

我们的实验内容是在存在运动点的情况下的一类标签排布的更新问题。

标签的自动排布问题在地图生成，地理信息系统和计算机图形学中是一个重要的问题。其中最基本的问题，就是对地图中的每个点、曲线、甚至区域添加一个标签（文字条，一般为矩形）。每个标签都必须与自己对应的标定物体（点、线、区域）相交，而标签之间不能重叠。其中点标签的排布问题是研究的一个重点[1]。

我们的实验问题，是在原有的点标签的排布问题上，引入了一个路径未知的运动点（在实际应用中，它可能是运动的人，车辆，或者鼠标），在运动点经过地图的过程中，同样也不希望被标签所遮盖，同时由于运动点的路径是未知的，需要通过即时的获得运动点的位置，并对各个标签进行响应更新，因此算法的效率就显得较为重要。

这一问题的应用可以包含运动路线的模拟显示，电脑游戏等等。

# 2 问题描述

在平面内给一组与坐标轴平行的标签（以及其标定点）组成的一个合法地图标定。一个点状的物体在平面内任意的运动，运动路径是不固定的，但是在点在沿着路径移动的过程中可以即时的将其所在位置通知给各个标定物体。

要求设计一个算法，使得在收到点所在位置的通知后，能够快速有效的更新各个标签的位置以避让运动点。在避让运动点的同时，仍然要保证每一个标签互不相交。在这一实验中，我们假设标签均为单位正方形，且标定点落在标签的某条边的中点上。

在这里，快速有效的更新定义为，尽可能的操作。操作包含：

1. 对标签进行翻转；
2. 尽量接近单位正方形的前提下缩放标签的大小。

对于这一问题，已经有一个对每一步更新复杂度为 O（nlogn）的算法，这一算法基于 2-SAT 问题的解，他并不利用当前标签排布的任何信息 [2]。

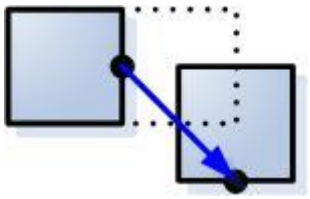
而在[3]中给出了一个新的算法，他在运动点开始汇报位置之前需要一个 O（$n^2$）的预处理过程，但是对于之后每步更新则只需要 O（logn+k）的复杂度（其中 k 是更新所包含的

操作数）。

# 3 算法描述

## 3.1 概念引入

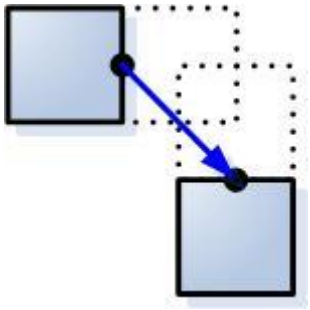
1. 多米诺边（domino edge）



图表 1 Domino edge

如上图，定义 Label A 为 la，其翻转为 flip（la）。如果 flip（la）与 lb 产生交集，则从 la 的固定点到 lb 的固定点连接的有向边称为 domino 边。

2. 阻碍边（blocking edge）

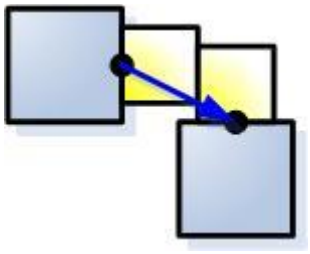


图表 2 Blocking edge

如上图，如果 flip（la）和 flip（lb）有交集，则从 la 的固定点到 lb 的固定点连接的有向边称为 blocking 边。

3. 边权重（edge weight）

当 la 和 lb，或者其翻转，之间产生了交集，则需要使用一个 resize 的步骤来避免，这个 resize 是有很多选择的。



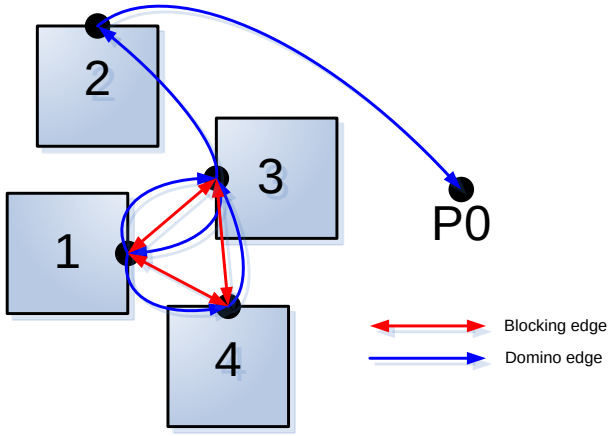
图表 3 Edge weight

我们设定 la resize 成边长 ra，lb resize 成边长 rb，则 min{ra，rb}中的最大值定义为 g(la，lb)。如上图，翻转之后的 label（黄色背景）大小就是这条 blocking 边的权值，那么边权重 w（e）定义为：

(1) w（e）=g（flip（la)，lb)，如果 e 是 domino 边

(2) w（e）=g（flip（la)，flip（lb))，如果 e 是 blocking 边

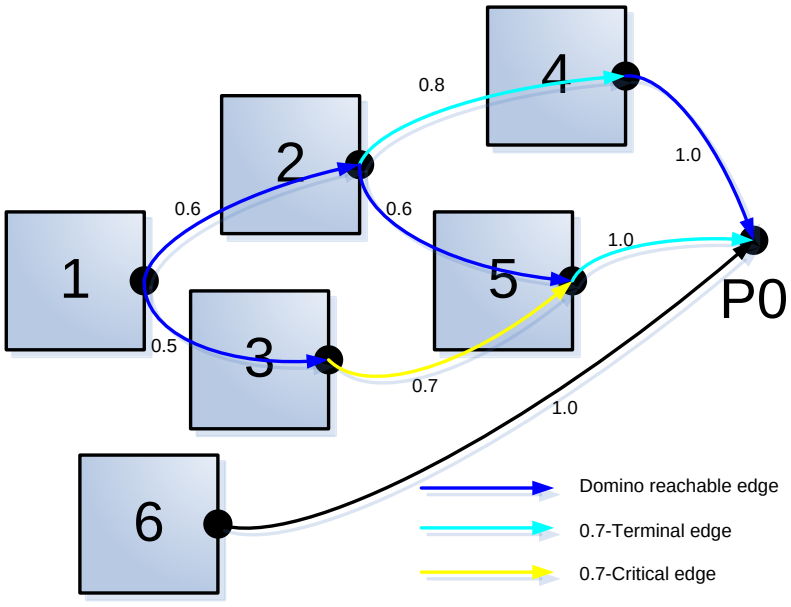
4. 冲突图（conflict graph）




图表 4 Conflict graph

在上图中所有的 label 都会引出一定数目的 domino 边和 blocking 边，如果一个 label 没有引出 domino 边，则从它引出一条权值为 1（单位权值）的 domino 边，指向一个虚点 p0。这样就能构造一个有向带权边图，每条边就按照上面所说定义为多米诺边和阻碍边，权值为相应的边权重。最终，这个有向带权边图就成为 conflict graph。见上图。

5. 可到达的 domino 边



图表 5 Domino-reachable edge

如果从 label q 可以经过若干条 domino 边到达 label A，那么从 label A 引出的任何 domino 边都称为从 label q 出发的可到达的 domino 边。在上图中，除了从 label 6 引出的指向虚拟点 P0 的 domino 边（标记为黑色）之外都是从 label 1 出发的可到达的 domino 边。

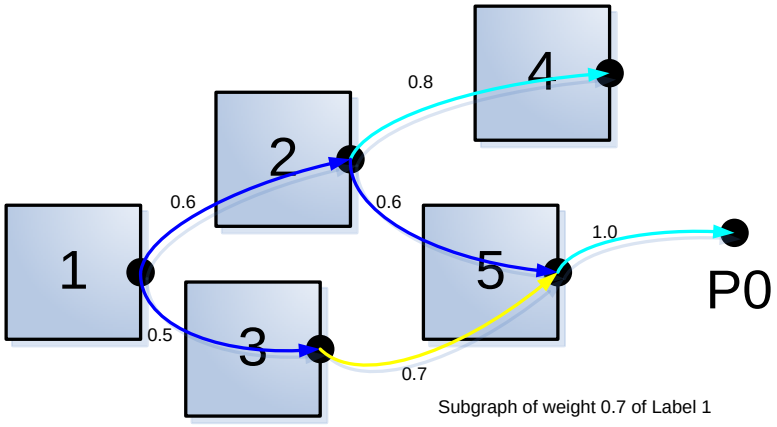
6. α－终止边

如果 a 是从 label q 出发的由 label A 引出的可到达 domino 边，而且从 label q 到 label A 的 domino 边权值均小于 α，a 的权值大于或者等于 α，那么 a 称为从 q 出发的 α －终止边。从 label q 到最后这个 label 的 domino 边序列称为从 label q 出发的 α －终止路径。在上图中，从 label 3 指向 label 5 的 domino 边（标记为黄色），从 label 2 指向 label 4 的 domino 边和从 label 5 指向虚拟点 P0 的 domino 边（标记为天蓝色）的这 3 条边称为从 label 1 出发的 0.7－终止边。

7. α－临界边

如果 a 是从 label q 出发的 α －终止边，且 a 的权值刚好等于 α，则 a 称为从 label q 出发的 α －临界边。见前面那张图中的从 label 3 指向 label 5 的 domino 边（标记为黄色）。

8. 权重子图



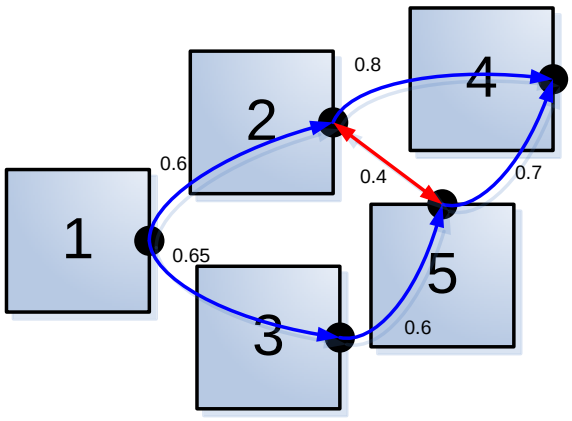

图表 6 Subgraph

在冲突图中提取出所有包含了从 Label q 出发的所有 α －终止路径的子图称为 q 的 α －权重子图，记做 $G_q^\alpha$。这个子图明显是唯一的。对于在子图中所有引出了 domino 边的 label，记做子图的内部 label，那么当内部 label 之间的 blocking 边的权重大于或者等于 α 时，我们称这个权重子图是有效的。上图就是 $G_1^{0.7}$。

9. 权重子图的有效性

如果一个权重为 α 的权重子图内部出现的所有 label 之间的 blocking 边的权值都不小于 α，那么我们称这个权重子图是有效的，如上图的 $G_1^{0.7}$ 就是有效的，但是见下图也是 $G_1^{0.7}$，

但是 label 2 和 label 5 之间出现了 blocking 边，且权值为 0.4，比 0.7 小，因此是无效的。




图表 7 Invalid Subgraph

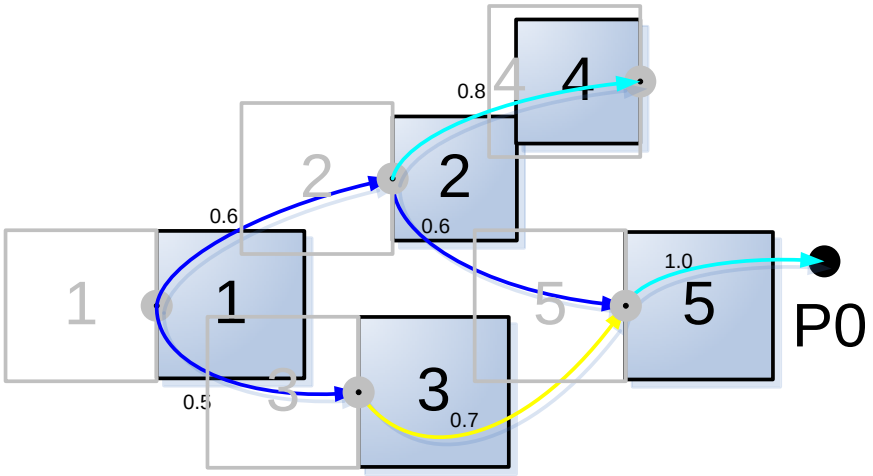
## 3.2 基本理论

1. 如果 $\alpha \le \beta$，那么有 $G_q^\alpha \subseteq G_q^\beta$

因为这种情况下，从 label q 出发的 β 一路径必然包含或者等于一条从 label q 出发的 α 一路径，因此明显成立。

2. 有效的 $G_q^\alpha$ 对应了一种从 label q 开始的翻转方案，且其 resize 代价为 α

对 $G_q^\alpha$ 中每条 α 一终止边，可以令其中的内部 label 都翻转，考虑到 $G_q^\alpha$ 的有效性，所以受内部 label 的 blocking 边权值影响，resize 代价最多为 α，而非内部 label 处于 α 一终止边的终点，所以 resize 代价也会大于或者等于 α。相反，任何一种从 label q 开始的最优代价为 α 的翻转方案，也对应了这个 $G_q^\alpha$。




图表 8 Subgraph 和翻转方案的对应

上图就是对应 $G_1^{0.7}$ 的翻转方法，当然，我们可以看出其实 label 4 还可以继续翻转，使 label 2 不需要 resize，但是这个就是更大的权值对应的翻转方案了。

## 3.3 算法概述

1. 预处理过程

   (1) 构造冲突图

   (2) 计算边权重

   (3) 按照边权重从小到大排序，获得一个排序后的边列表

   (4) 构造点和 Label 初始位置的数据结构

   (5) 计算每个 Label 的翻转区域

预处理过程需要 O（$n^2$）的时间：构造冲突图可以用一条竖直扫描线来对全图进行扫描，这个过程需要 O（nlogn）的时间；每条边的边权重只需要 O（1）的时间就可以计算出来；排序和数据结构的构造也只需要 O（nlogn）的时间；最后一步对每个 Label 都需要 O（n）的时间。因此总共的时间消耗为 O（$n^2$）。

2. 更新算法

   (1) 找到移动点 q 所在的 Label。

   (2) 如果不存在，则算法结束，不需要挪动或者改变大小。

   (3) 如果 q 不在这个 Label 的翻转区域，则修改 Label 的大小来躲避 q。结束。

   (4) 以冲突图和 Label 作为输入，寻找最大的 α 和子图，使这个子图中所有边权重都大于或者等于 α 。算法结束。

算法的最后一步需要 O（logn＋k）的时间消耗。

## 3.4 冲突图

1. 数据结构

冲突图在后期构造权重子图中要使用到，其必须提供的功能是：

(1) 获得一个 label 对应的所有 domino 边。

直接使用一个链表数组来存储即可。数组以 label 的 ID 为索引，每个链表包含了指向的 label ID，domino 边的权重。

(2) 判断两个 label 之间是否有 blocking 边，并得到其权值。

最快捷的方式就是保存一个 n×n 的数组，但是因为实际情况中可能 blocking 边很少，所以还是使用 domino 边的存储方式。

(3) 从小到大依次获取边权值。

直接使用一个数组存储即可。

2. 构造算法

构造冲突图，最简单的方法就是对图中所有的 Label 进行两两比较，判断其是否会产生 domino 边或者 blocking 边。这种方案的时间开销是 $O(n^2)$。一个时间消耗少的方法是扫描线算法，可以避免对图中的每两个 Label 都需要比较。

由于 Label 的大小是固定的，考虑翻转之后，一个 Label 的影响范围也只有两倍 Label 边长而已。于是我们可以使用如下的简化方法：

(1) 对所有 Label 按照固定点的 x 坐标进行升序的排序，放入集合 G；

(2) 从 G 中取出 $Label_1$，加入集合 K；

(3) while G 非空

(4) 从 G 中取出下一个 label，记为 $Label_i$。

(5) For each $Label_j$ in 集合 K

(6) If $Label_i$ 和 $Label_j$ 的固定点的水平距离超过了两倍 Label 边长

(7) 从 K 中删去 $Label_j$，

(8) Else

(9) 计算两者的 domino 边和 blocking 边。

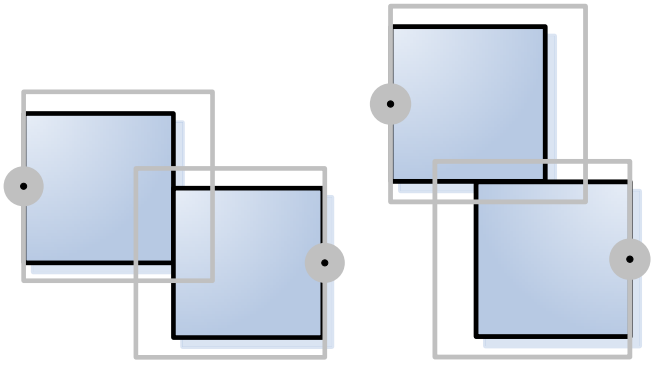
(10) 将 $Label_i$ 加入集合 K。

在最好的情况下，比如所有的 label 距离适当，则时间消耗为 $O(n)$，但是如果 label 相距非常近，则即使两两都没有 domino 边或者 blocking 边，也会需要消耗 $O(n^2)$ 的时间。

3. 边权重的计算

边权重的计算分为以下的情况：

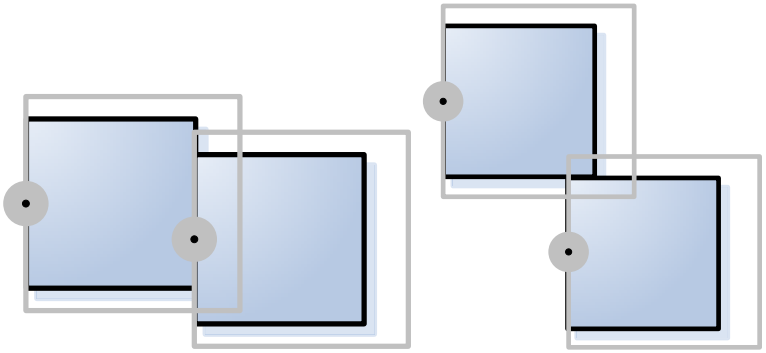
(1) 方向相对



图表 9 方向相对的 edge weight

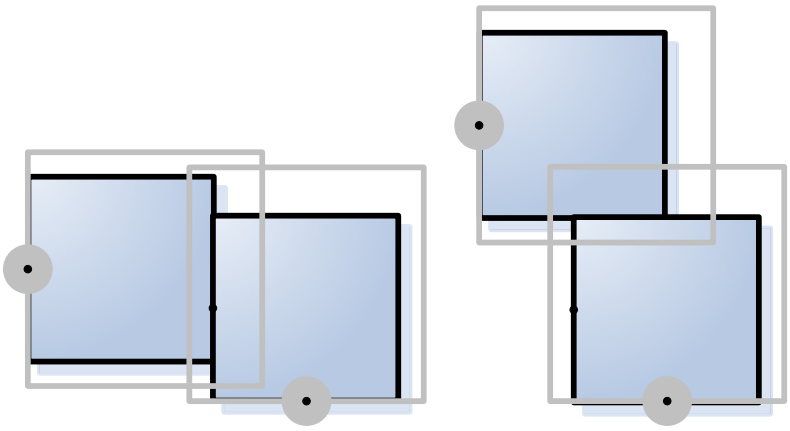
这两种情况需要根据固定点的 x，y 坐标的差异来计算边权重。

(2) 方向相同



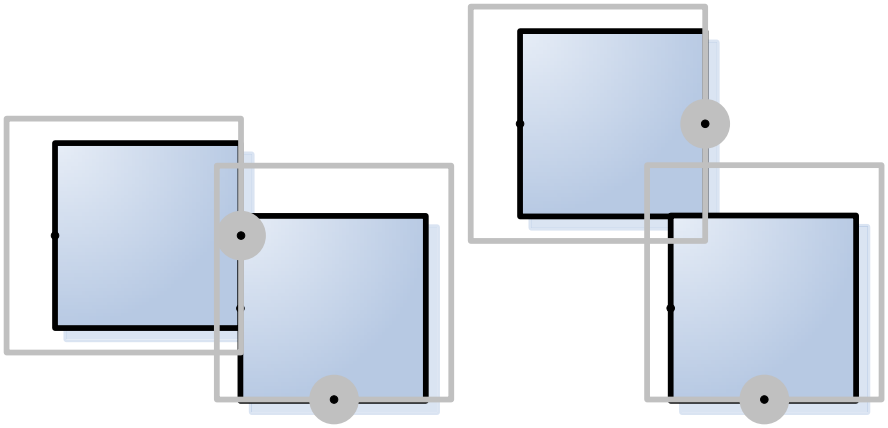
图表 10 方向相同的 edge weight

(3) 垂直相交且固定点射线相交



图表 11 垂直相交且固定点射线相交的 edge weight

(4) 垂直相交且固定点射线不相交



图表 12 垂直相交且固定点射线不相交的 edge weight

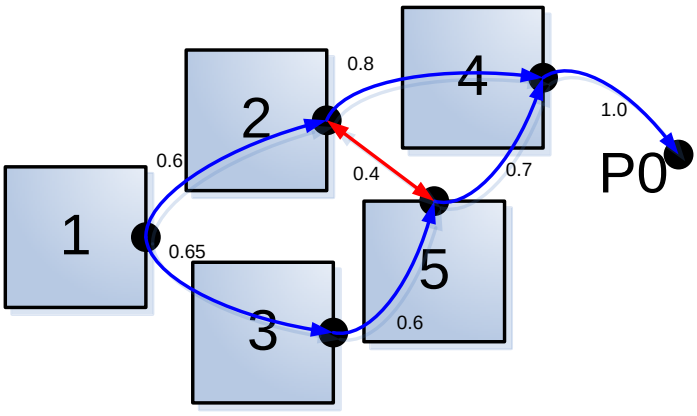
## 3.5 权重子图

1. 基本思路

根据前面的第二条基本理论，一个权重子图对应了一种翻转方案，且这个翻转方案的代价就等于这个权重子图的权重。因此我们要找到一个最好的方案，就是要找到一个权重最大的权重子图。

另一方面，根据第一条理论，我们可以在低权值的基础上逐步提高权值，就可以慢慢扩展整个图，得到最高的有效权重子图。

我们通过下面这个例子来看最大有效权重子图是如何构造的。下图是冲突图：

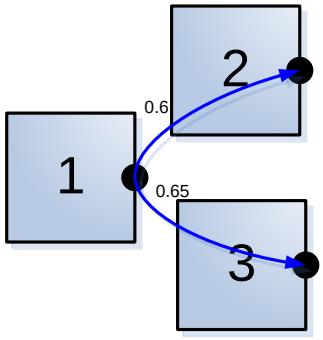



图表 13 冲突图示例

我们采用逐步提高阈值的方法来计算 label 1 的最大有效权重子图：

(1) $\alpha = 0.4$

从 label 1 开始扩展 domino 边，知道权值大于或者等于 0.4，得到的权重子图如下：



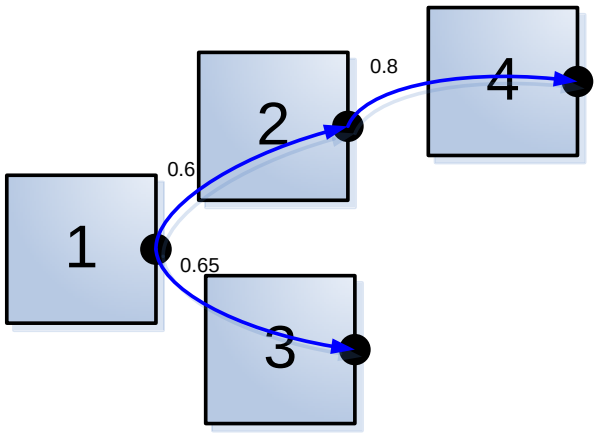

图表 14 扩展权值为 0.4 的 Subgraph

(2) α =0.6

由于上次扩展出来的 domino 边的权值已经超过了这次的权值，所以这次的权重子图没有变化。

(3) α =0.65

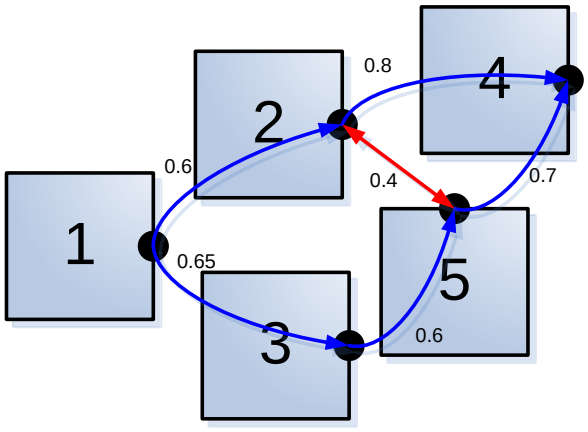
从 label 1 指向 label 2 的 domino 边已经不够这个权值了，所以从 label 2 开始向后扩展，得到的权重子图如下：




图表 15 扩展权值为 0.65 的 Subgraph

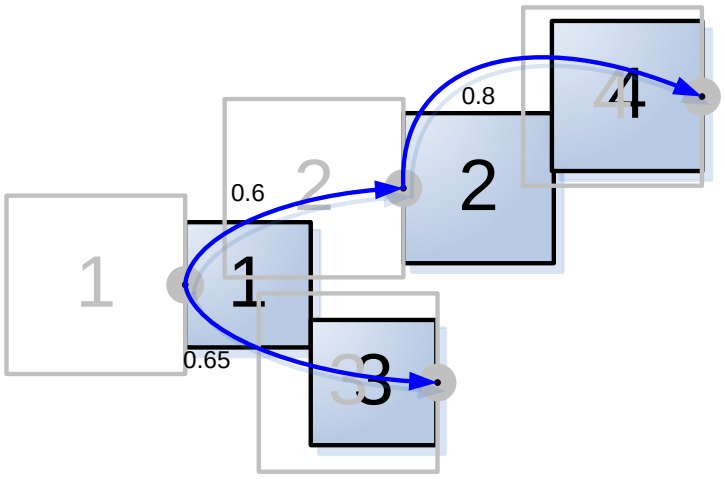
(4) α =0.7

这次需要扩展 label 3 了，如下图：



图表 16 扩展权值为 0.7 的 Subgraph

但是前面已经说过了，这个权重子图是无效的权重子图，所以不能继续扩展了。因此算法到此为止，最后的结果是应该采用权重为 0.65 的子图对应的翻转方案，如下图：



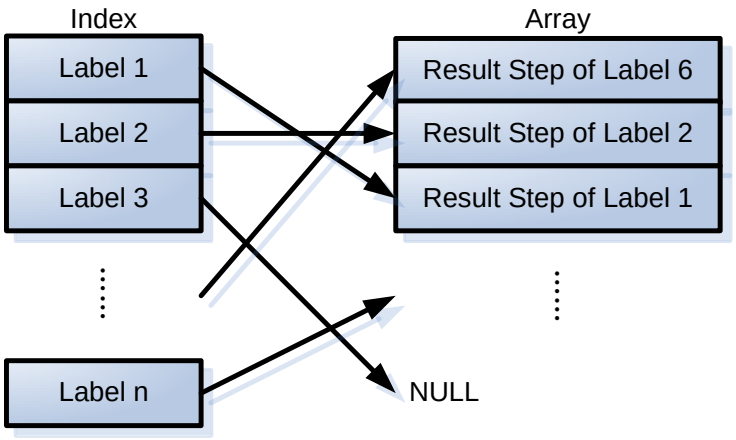

图表 17 对应的翻转方案

其中，label 1 和 label 3 会 resize 到 0.65，label 2 和 label 4 会 resize 到 0.8。所以这个翻转方案的代价是 0.65。我们可以看到，如果 label 4 继续翻转，那么 label 2 和 label 4 都是不需要 resize 的，但是这个已经不会影响最后的代价了，所以步骤越少越好。

2. 数据结构

首先为了能够实现单步，那么需要记录每一步的 label 翻转和 resize。因此每一步要记录 label ID，是否翻转，resize 的值，我们用一个类 ResultStep 来记录。另外，每次按照一个权值扩展之后，还需要把这次扩展出来的所有步骤统一返回给界面，需要一个 ResultStep 的数组，另外加上本次扩展的权值和总数目，用类 ResultPart 来记录。

上面的数据是用于单步调试使用的，因为一个 label 可能在整个算法中有多次动作，但是最后输出结果的时候只需要关心最后的状态；另一方面，由于算法扩展到最后一般都是扩展出无效的权重子图才停止，所以我们不能没扩展一步就把这个步骤加入最终的结果中，而是需要在每次需要提高权值扩展新扩展出一批的时候才加入。因此最终输出的数据结构不能使用 ResultPart，而是需要在 ResultPart 的基础上记录一些新的信息，我们使用一个类 LabelResult 来记录，见下图：




图表 18 LabelResult

前面的具体步骤不使用数组来保存的目的就是为了节省空间，因为大部分 label 的权重子图并不会需要所有 label 都有动作，因此最后使用一个 hash 数组来完成。

除了最终返回的结果之外，在算法的扩展步骤中，我们需要记录一些信息：

(1) 目前已经访问过的 label

(2) 所有的终止 label

(3) 所有的临界 label

上面的所有内容都即有遍历的需求，也有判断的需求，所以不但存储了相应的 label ID，还需要存储标志位，表示一个 label 属于哪种情况。

3. 基本流程

权重子图构造算法。Input：触发的 label ID i，冲突图。Output：LabelResult。

(1) 初始化临界 label 的集合 A，终止 label 的集合 B，翻转的 label 集合 C。

(2) 初始化中间结果和最终结果

(3) 将 i 加入 A

(4) For each 权值 w（按照从小到大排序） in 冲突图

(5) While A 不为空

(6) 从 A 中取出 label，ID 为 j。

(7) 扩展出所有的 w一终止边。

(8) If 当前权重子图无效

(9) 返回最终结果

(10) 判断原来的终止 label 是否变成临界 label 了

(11) 将这次扩展的内部 label 加入 C，终止 label 加入 B，临界 label 加入 A。

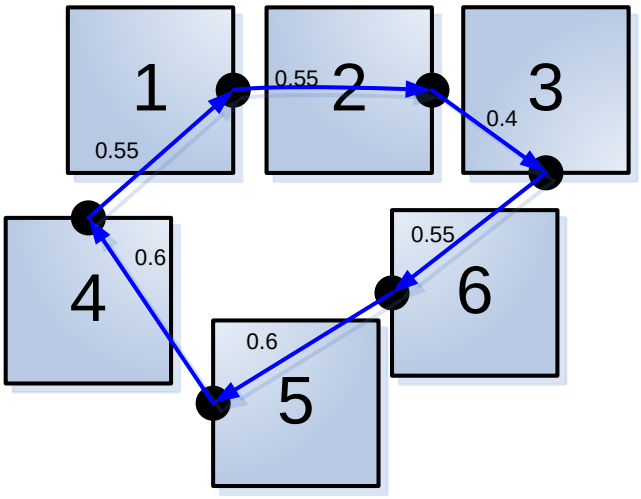
(12) 将这次的中间结果加入最终结果。

4. 注意事项

(1) 多次扩展

一个 label 如果有多个 domino 边指向它，那么在构造权重子图的过程中可能就会出现被多次访问到的情况。如下面的冲突图，我们在扩展 label 1 的权重子图过程中最终会扩展到 label 4，但是 label 4 指向的 label 1 已经被扩展了，这个时候我们应该按照新的权值来扩展 label 1 还是不需要扩展 label 1 了呢？

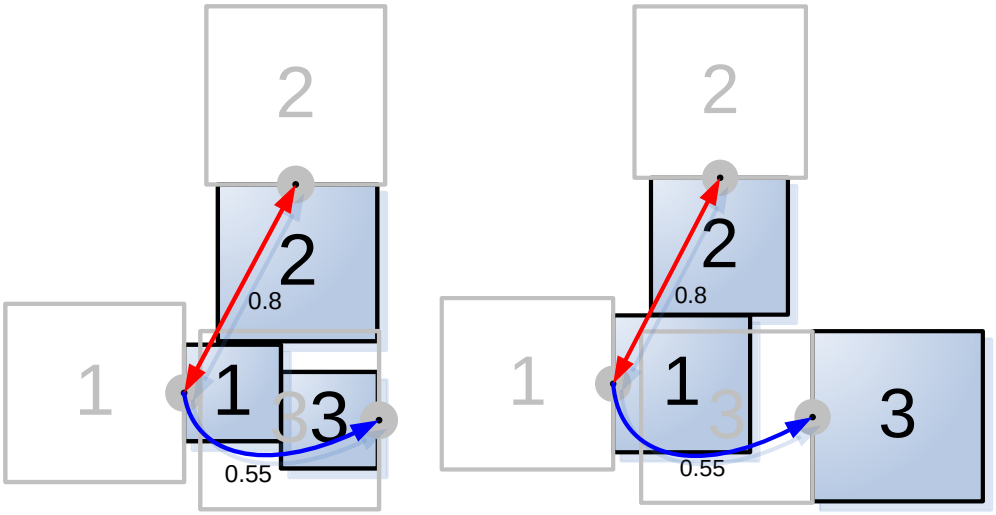
我们可以考虑一下，如果我们要扩展 label 1，说明 label 1 必须要引出不会小于从 label 4 指向 label 1 的 domino 边权值的 domino 边。但是明显，如果有这样的 domino 边，那么在最初扩展 label 1 的时候就肯定已经被扩展过了。因此。Label 1 不需要再次被扩展。而 label 4 还是按照要求翻转。




图表 19 多次扩展(环状冲突图示例)

(2) 权值返修改

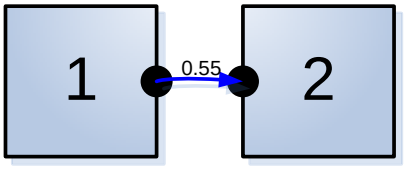
权值返修改是指，一个 label 受到其它 label 的影响，造成它 resize 的大小需要改变，比如下图中扩展 label 3 的时候，虽然不会影响到 label 2，但是因为从 label 1 有指向 label 3 的 domino 边，且 label 1 和 label 2 之间有 blocking 边，因此，对 label 1 和 label 2 都要重新调整大小。




图表 20 权值返修改

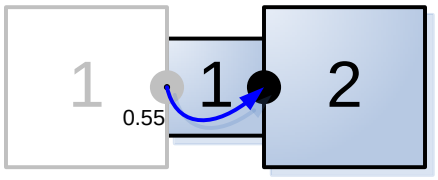
(3) 不对称调整大小

Domino 边和 blocking 边有时候不是需要调整两个 label 到同样的大小。比如下图：



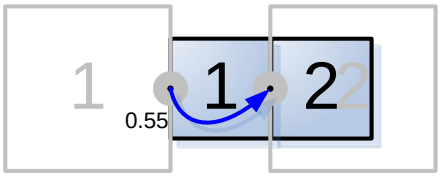

图表 21 不对称 domino edge

对于 label 1，它的理想调整方法见下图：




图表 22 不对称 resize 方案

因此实际上调整方案应该是不对称的，但是由于在冲突图计算时没有考虑到上面的情况，因此调整方案变成了下图所示：




图表 23 对称 resize 方案

## 3.6 Flipping Region

在预处理过程中，需要针对每个 Label 计算计算其对应的 Flipping Region，以便在处

理过程中快速判断被 p 点盖住的 Label 是否需要翻转。[3]中对于 Flipping Region 的计算非常不明确，本节中介绍算法中计算 Flipping Region 的详细步骤。

# 1. 定义及符号约定

**定义 Flipping Region** 当前问题中对于每个 Label 都有对应的 Flipping Region，指的是 Label 内部的一个多边形区域。若躲避点处于多边形区域内时，当前 Label 应该采用翻转操作；而躲避点处于 Label 内部但在 Flipping Region 外时，当前 Label 应当调整大小以避开躲避点。

**定义 Flipping Region Segment** 当前问题中 Label 的 Flipping Region 由数段不同的线段拼接而成多边形区域，其中一部分是其他 Label 经过翻转调整大小后和当前 Label 相交贡献得到的，这些线段称为 Flipping Region Segment.

为方便描述，定义如下符号：

对于当前正在处理，即需要计算其 Flipping Region 的 Label 记为 $L_0$；

不考虑输入点，翻转 $L_0$，能够保证多边形区域之间不相交的而得到的最大 Label 边长记为 $\alpha$，如上所述 $\alpha$ 可以通过利用计算 $G_q^\alpha$ 得到；

调整 $L_0$ 的边长为 $\alpha$ 后得到的 Label 称为 $L_\alpha^{(0)}$；

经过翻转调整大小后可能与 $L_0$ 相交产生 Flipping Region Segment 的 Label 记为 $L$，易知 $L_0 \neq L$；

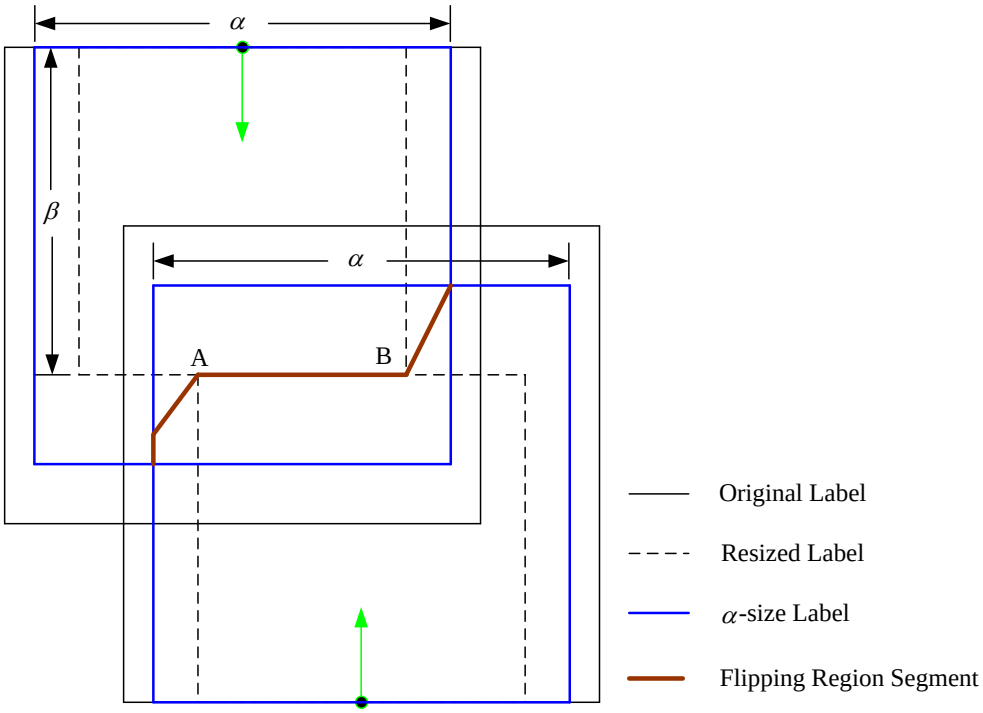
对 $L$ 翻转后得到的 Label 记为 $L_f$；

调整 $L_f$ 的边长为 $\alpha$ 后得到的 Label 称为 $L_\alpha$；

若存在 $L$ 到 $L_0$ 的 Domino Edge，则边长记为 $\beta$；

调整 $L_f$ 的边长为 $\beta$ 后得到的 Label 称为 $L_\beta$；

给定 $L_0$ 内部的一点 $p$，调整 $L_0$ 的大小避开 $p$ 点的代价为 $f_r(p)$，翻转 $L_0$ 避开 $p$ 点的代价为 $f_f(p)$。




图表 24 Flipping Region Segment ($\alpha > \beta$)

对于上图所示情况，$L_0$ 为左上方实线边 Label，$L_f$ 为右下方实线 Label，蓝色 Label 为 $L_\alpha$，和实心电相接的箭头表示。虚线构成的两个 Label 表示 $L_0$ 和 $L_f$ 的大小都调整为 $\beta$ 时，两 Label 不相交。图中深红色构成的线段表示 $L$ 与 $L_0$ 形成的 Flipping Region Segment。

***注：本章图中都采用同样的颜色标记。***

## 2. 计算 Flipping Region Segment

后面我们会看到，Flipping Region 是由部分 $L_\alpha^{(0)}$ 和多段不相交的 Flipping Region Segment 拼接而成的，我们首先讨论如何计算两个多边形相交形成的 Flipping Region Segment，之后再讨论如何将这些 Flipping Region 拼接起来。下面的讨论都是基于只有一个正方形 $L$ 和 $L_0$ 相交。

我们首先针对上图中给定的情况讨论 Flipping Region Segment 的成因以及影响它的各方面原因。我们可以利用之前的计算确定 $\alpha$，其含义为翻转 $L_0$ 后为了确保 Label 之间不相交，对对于其他 Label 进行翻转和调整大小操作后能得到的最小边长的最大值。对于上图给定的情况，给定 $L_0$ 内部的某个输入点 $p$（注意这里所说的内部与外部包括边界上的点，边界上的点如何翻转会在后面讨论）

(1) 若 $p$ 在 $L_\alpha^{(0)}$ 的外部，此时 $f_r(p)\geq\alpha$， $f_f(p)\leq\alpha$，应该对 $L_0$ 调整大小；

(2) 若该点在 $L_\alpha^{(0)}$ 的内部且在 $L_\alpha$ 的外部，此时有 $f_r(p)\leq\alpha$，$f_f(p)\geq\alpha$，应该翻转 $L_0$；

(3) 若 $p$ 在 $L_\alpha^{(0)}$ 的内部且在 $L_\beta$ 的内部，此时有 $f_r(p)\geq\beta$， $f_f(p)\leq\beta$，调整 $L_0$ 的大小；

(4) 若 $p$ 在 $L_\alpha^{(0)}$ 的内部且在 $L_\beta$ 的外部、 $L_\alpha$ 的内部时，注意到线段 AB 上的点确保 $f_r(p)=f_f(p)=\beta$，同样的对于 A 点朝左下角继续延伸，对于 B 点朝右上角继续延伸，直到 $L_\alpha$ 的边界，同样的保持线段上的点确保 $f_r(p)=f_f(p)=\beta$。在整条线段的左上方都有 $f_r(p)<f_f(p)$ 对应 $L_0$ 翻转的情况，右下方的区域对应 $f_r(p)>f_f(p)$ 对应 $L_0$ 调整大小的情况。

观察到上述情况中可以发现，Flipping Region Segment 就是 $f_r(p)=f_f(p)$ 的点构成的线段。事实上整个系统是一个线性系统， $f_r(p)=f_f(p)$ 形成了一个闭合的多边形区域，Flipping Region Segment 就是那些不在 $L_\alpha^{(0)}$ 边界上的线段。

我们可以看到 Flipping Region Segment 受以下因素的影响：

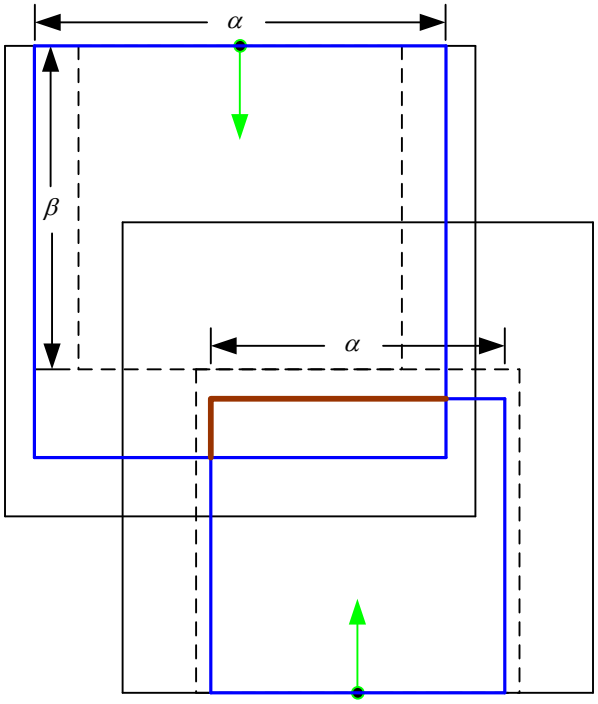
（1） $L_0$ 与 $L$ 的位置朝向关系；

（2）由 $L_0$ 与 $L$ 的位置朝向关系决定的 $\beta$；

（3）由整体排列情况决定的 $\alpha$；

其中 $\beta$ 主要体现 $L_0$ 与 $L$ 之间的关系， $\alpha$ 体现了其他 Label 对 $L_0$ 的影响。

特别的我们还要讨论另外一种情况， $\alpha\leq\beta$，如下图所示，




图表 25 Flipping Region Segment ($\alpha \le \beta$)

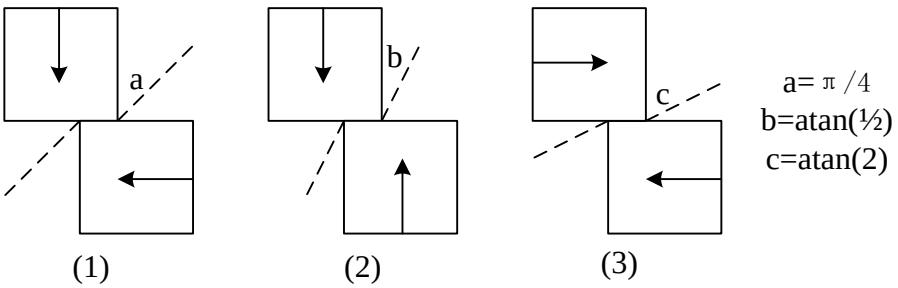
和上面的分析类似，这时 $L_\alpha$ 边界在 $L_\alpha^{(0)}$ 内部的部分就形成了 Flipping Region Segment。

由此我们得到 Flipping Region Segment 的计算方法：

(1) $\alpha \le \beta$，计算 $L_\alpha$ 边界在在 $L_\alpha^{(0)}$ 内部的部分，确定 Flipping Region Segment；

(2) $\alpha > \beta$，计算 $L_\beta$和$L_\beta^0$ 相交线段，从线段端点向外延伸直到 $L_\alpha^{(0)}$ 的边界为止。

这里特别需要注意的是，在情况（2）中从线段端点开始延伸方向并不是固定的，如下图所示，




图表 26 线段 AB 扩展方向

其中，从两个端点的扩展方向相差 π，

（1）$L_0$ 与 $L$ 方向不在同一水平轴上时，角度a为 π /4；

（2）$L_0$ 与 $L$ 方向在同一水平轴上但与线段AB不在同一水平线上时，角度b为 $\arctan\frac{1}{2}$；

（3）$L_0$ 与 $L$ 方向在同一水平轴上但与线段AB在同一水平线上时，角度b为 $\arctan 2$；

从上面的分析可以看出，两个 Label 排列方式多种多样，相交的方式多种多样，两个多边形之间 Domino Edge 边长和 $\alpha$ 的大小关系也会不同，这就决定了 Flipping Region Segment 多种多样。根据 Flipping Region Segment 的产生方式进行分类，才能行之有效地进行计算，简化算法的设计难度。

算法中，对 Flipping Region Segment 的分类顺序如下：

(1) 判断 $L_\alpha$ 与 $L_\alpha^{(0)}$ 是否相交；

(2) 判断 $\alpha$ 和 $\beta$ 的大小关系；

(3) 计算 $L_\beta$ 与 $L_\beta^{(0)}$ 交线位置 AB；

(4) 根据 $L_0$ 与 $L$ 的相对位置判断 AB 延长线方向；

(5) 根据 $L_0$ 与 $L$ 的相对位置判断 AB 延长线可能和 $L_\alpha^{(0)}$ 的那些边相交；

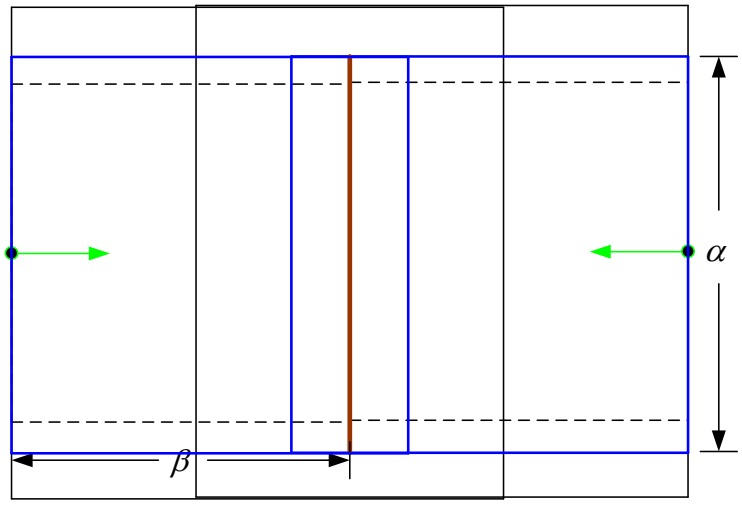
(6) 计算延长线交点位置；

# 3. Flipping Region Segment 退化情况

计算 Flipping Region Segment 的过程中会出现三种退化情况，我们需要针对这三种情况分别处理。

**退化情况 1**

如下图所示，当 $L_\alpha$ 与 $L_\alpha^{(0)}$ 两个正方形正对时，这时由 A、B 顶点发出的延长线不再是上面所给出的三种情况之一，而是一条与两 Label 固定点连线垂直方向的射线。我们可以根据两个 Label 固定点的位置是否基本在同一和坐标轴平行的直线上判断是否出现了这种情况；对于这种情况，我们单独处理由 A、B 发出的射线的角度。

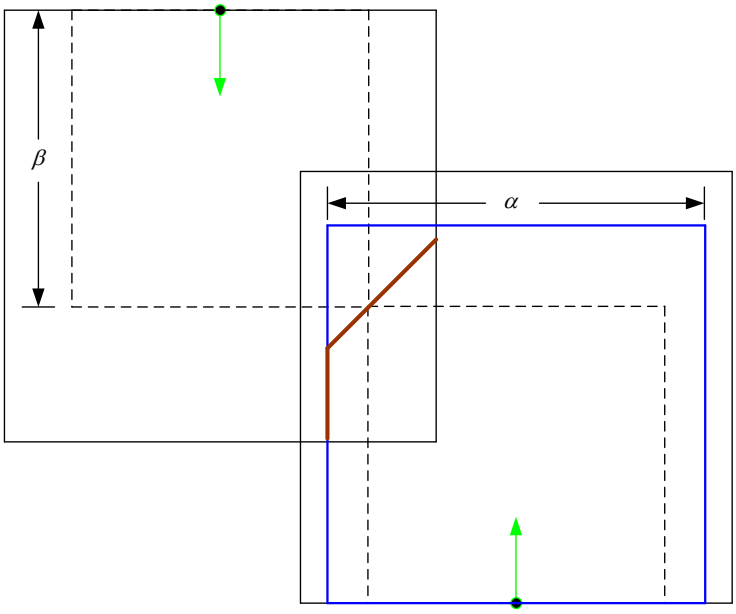



图表 27 Flipping Region Segment 退化情况 1

**退化情况 2**

当两个正方形右顶点斜交时，这时 A、B 两点退化为一点，也就是说 Flipping Region 上会出现间距为 0 的两点。这会对后来的点定位过程产生影响，我们的处理方法就是在点定位

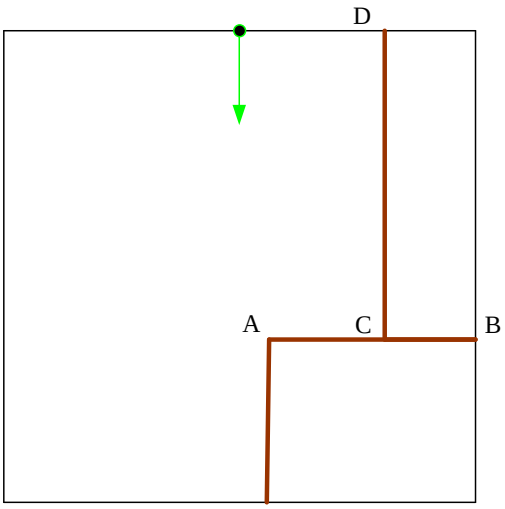
的过程中，如果出现间距为 0 的两点，将忽略这条边。




图表 28 Flipping Region Segment 退化情况 2

**退化情况 3**

两个多边形在 $L_\alpha^{(0)}$ 分别相交产生 Flipping Region Segment，但是这两个 Flipping Region Segment 的边相邻，如下图所示，两个 Label 分别产生了 AB、BCD 这样的 Flipping Region Segment，这就意味着最后的 Flipping Region 会产生一个面积为 0 的条带，这并不会影响最后的点定位过程，程序中并没有对这种情况加以特殊处理。




图表 29 Flipping Region Segment 退化情况 3

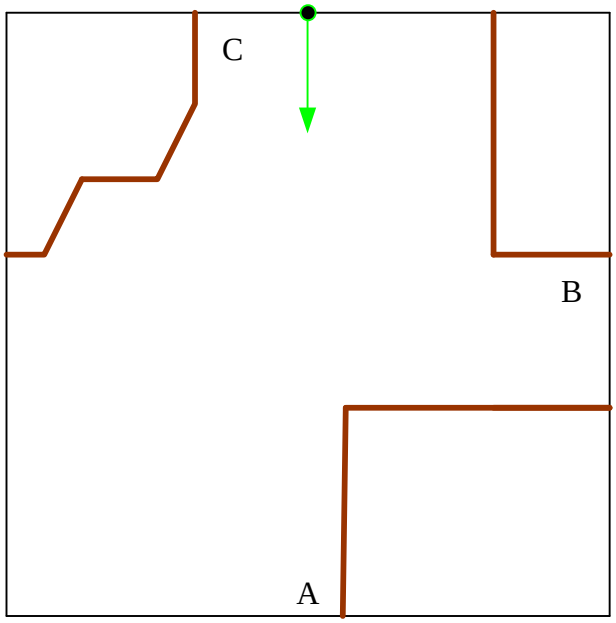
# 4. Flipping Region Segment 合并

**定理 Flipping Region 是由多段不相交的 Flipping Region Segment 和 $L_\alpha^{(0)}$ 边界拼接而成拼接而成的。**

**证明** 上面分析 Flipping Region 的产生原因可以知道 Flipping Region 是由多个 Flipping Region Segment 和 $L_\alpha^{(0)}$ 的边界拼接而成，这里需要说明的是不同 Label 和 $L_\alpha^{(0)}$ 相交产生的 Flipping Region Segment 不相交就可以了。如果不是这样的话，就意味着整个躲避方案中最大的 Label 边长小于 $\alpha$，这与 $\alpha$ 的定义矛盾。

既然 Flipping Region Segment 之间不相交，计算 Flipping Region 就是一个将这些线段合并的过程。将每一段 Flipping Region Segment 按照其起点排列在整个 $L_\alpha^{(0)}$ 之上，然后从某一个 Flipping Region 的起点开始顺序遍历就会得到整个 Flipping Region 的顶点序列。

Flipping Region 的顶点序列返回给界面用于绘制 Flipping Region，也可以用于更进一步的点定位。



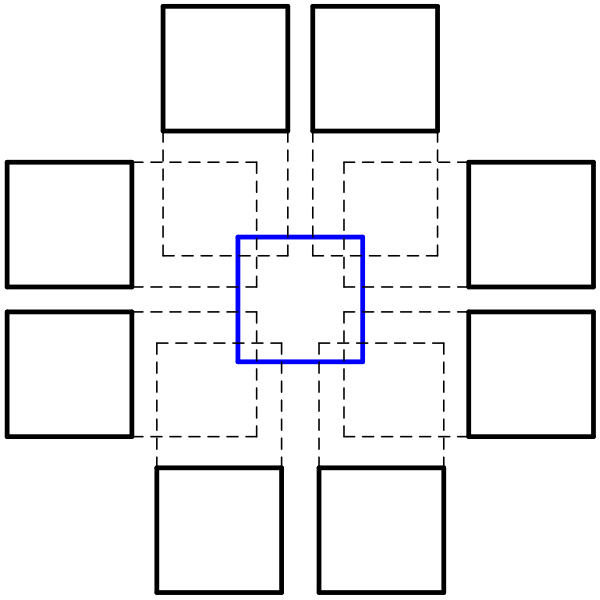

图表 30 Flipping Region Segment 合并

# 5. Flipping Region 复杂度分析

类似于[3]引理 5 的证明步骤，有如下定理：

**定理 每个 Label 至多被 8 条 Domino Edge 指向。**

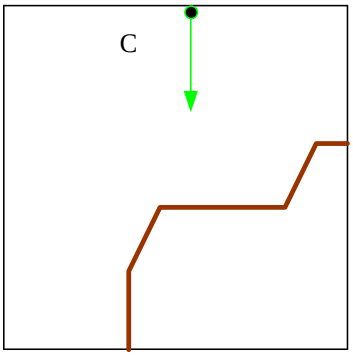
证明可以用下图说明，从图上可以看出，不可能有第九条指向中央 Label 的 Domino Edge。



图表 31 Label 最多被 8 条 Domino Edge 指向

**定理 两个 Label 产生 Flipping Region Segment 的线段条数为$O(1)$。**

如下图所示，两个 Label 相交而产生的 Flipping Region Segment 中线段条目数最多为 5。




图表 32 Flipping Region Segment 线段条数最多为 5

考虑计算每个多边形 Flipping Region 的时间复杂度，在已知$L_0$翻转后其他 Label 调整方案的情况下，对于每个多边形需要耗费$O(1)$的时间确定 Flipping Region。在这个过程中判断过程不需要存储额外的历史信息，所需要的空间复杂度为$O(1)$。

综上可知，单纯计算所有多边形的 Flipping Region 需要$O(n)$的时间，空间复杂度为$O(1)$。

## 3.7 Point Location

预处理结束后，算法开始接受输入点并给出 Label 的躲避方案。在这个步骤中需要确定输入点是否在某个 Label 内，并进一步确定是否在该 Label 的 Flipping Region 内，这是一个

Point Location 的过程。整个定位过程分为两步：

（1） 确定输入点所在的 Label 或者不在任何一个 Label 内；

（2） 若输入点处于某个 Label 内，判断输入点是否属于该 Label 的 Flipping Region；

对于（1）我们采用 Trapezoidal Map 算法，初始化过程采用 Randomized Incremental Algorithm。Trapezoidal Map 算法查找结构的构造在整体算法的初始化步骤中完成，期待时间复杂度为$O(n\log n)$，空间复杂度为$O(n)$。对于给定输入点，点定位的期待查询时间为$O(\log n)$。

为了提高算法效率，我们并没有将 Label 的四条边都加入 Trapezoidal Map 中。对于每一个 Label，我们仅将平行于 x 轴且 y 值较小的线段加入到梯形图中。点定位的结果返回给定点所在的梯形（算法中梯形平行边与 y 轴平行），确定梯形非平行边中 y 值较小的那条边所对应的 Label，根据 Label 的边长就可以直接判断该点是否在这个 Label 内。这个改进和当前问题中 Label 是正方形且边和坐标轴平行有关，类似的方法可以推广到矩形情况。

算法实现来自于[5]，我们改进了程序中对于边界情况处理不完善的问题。

对于（2），根据上节中定理可知每个 Flipping Region 的线段数上限是一个与 Label 个数 n 无关的常数。Flipping Region 中得到的边界点的顺序按照逆时针排序，我们只需要判断给定点在 Flipping Region 边界点顺序构成的线段左侧就可以了，这个操作的时间复杂度为$O(1)$。根据上一章的结果构造 Flipping Region 的时间复杂度为$O(n)$，保存 Flipping Region 的空间复杂度为$O(n)$。

特别需要注意在 Flipping Region 边界上的情况，边界上的点表示在这一点无论是对当前 Label 翻转还是调整大小最终方案得到的最小 Label 的大小是一样的。根据题目所描述的目标，应该调整当前 Label 的大小以减少操作的次数。

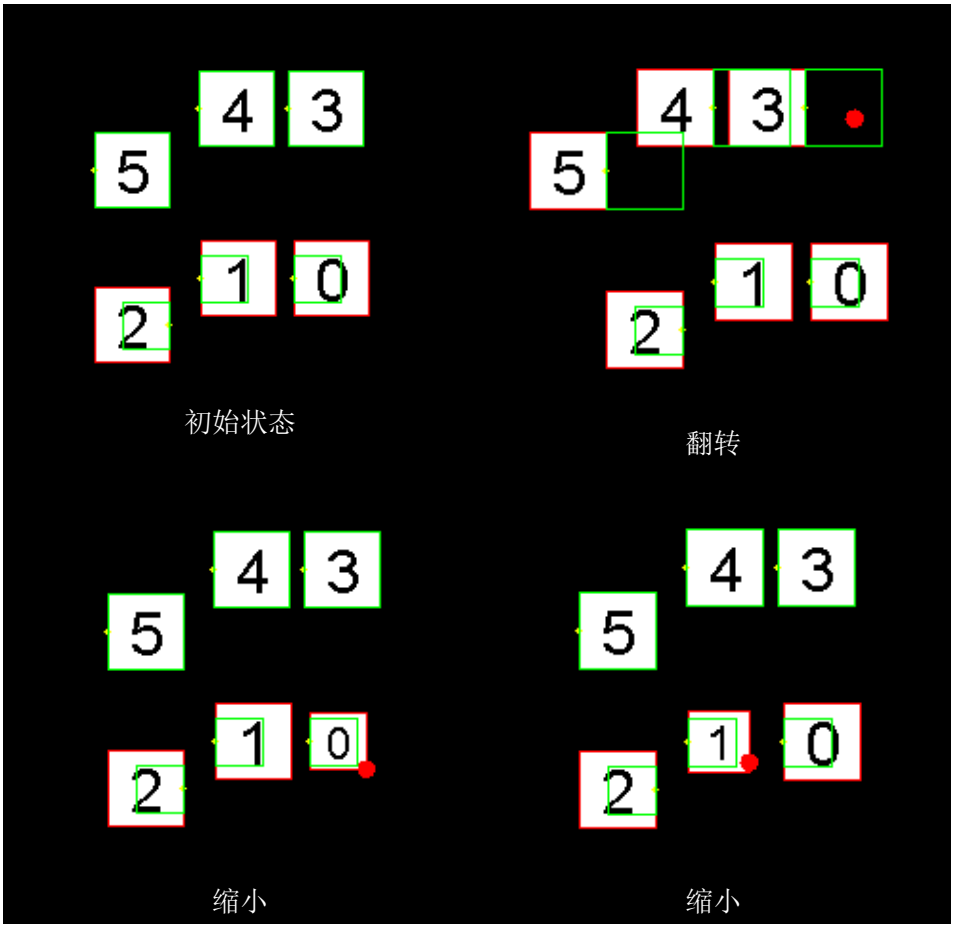
综上可知，完成定位目标（1）（2）预处理的时间复杂度为$O(n\log n)$，空间复杂度为$O(n)$，每次查询定位的期待时间为$O(\log n)$。

# 4 结果分析

## 4.1 正确性测试

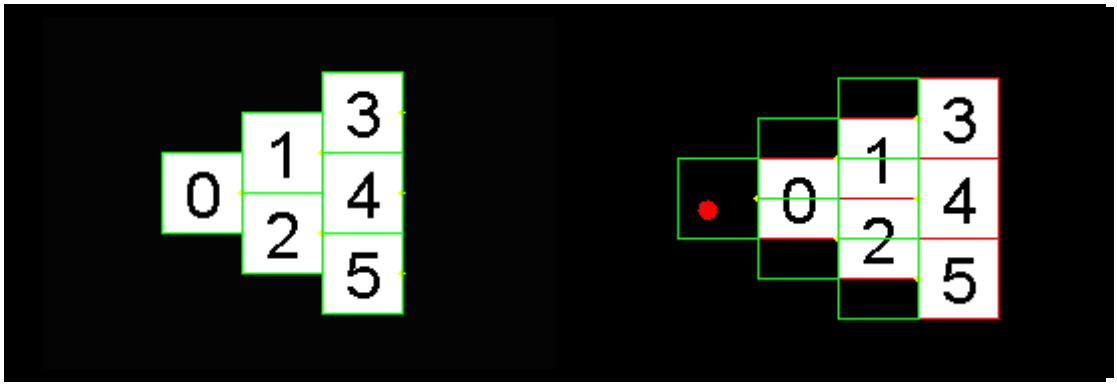
以下几个例子考察了程序在几种不同的基本情况下的正确性。

**最基本的翻转和缩小：**



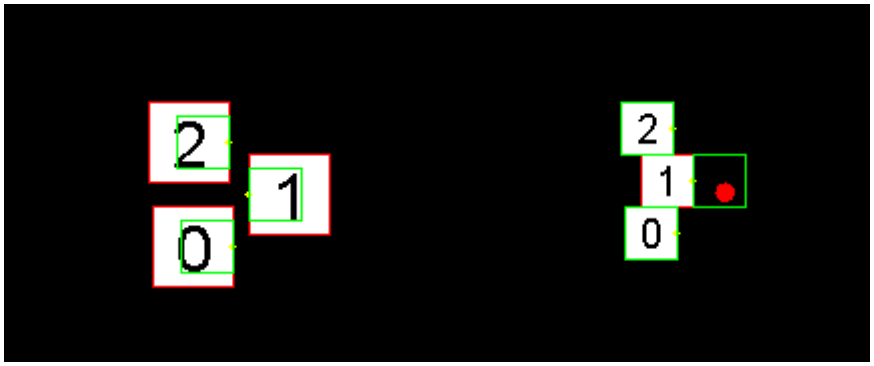

图表 33 测试样例-基本正确性测试--最基本的翻转和缩小

**全部翻转的情况：**



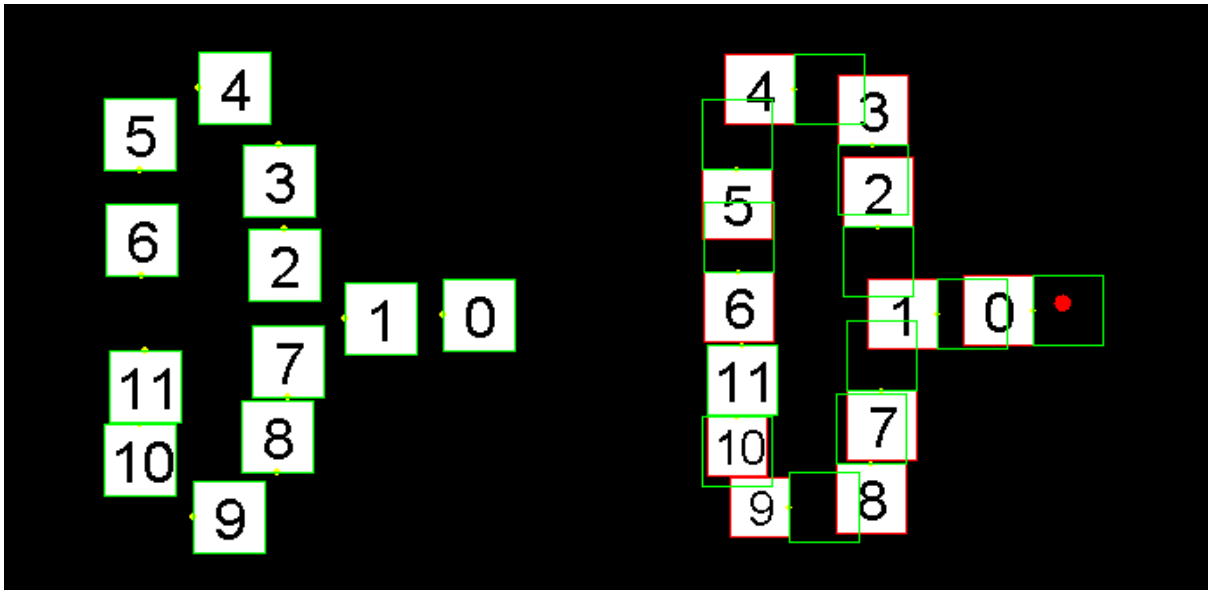

图表 34 测试样例-基本正确性测试--全部翻转

**全部缩小的情况：**




图表 35 测试样例-基本正确性测试-全部缩小

**翻转与缩小混合的情况：**



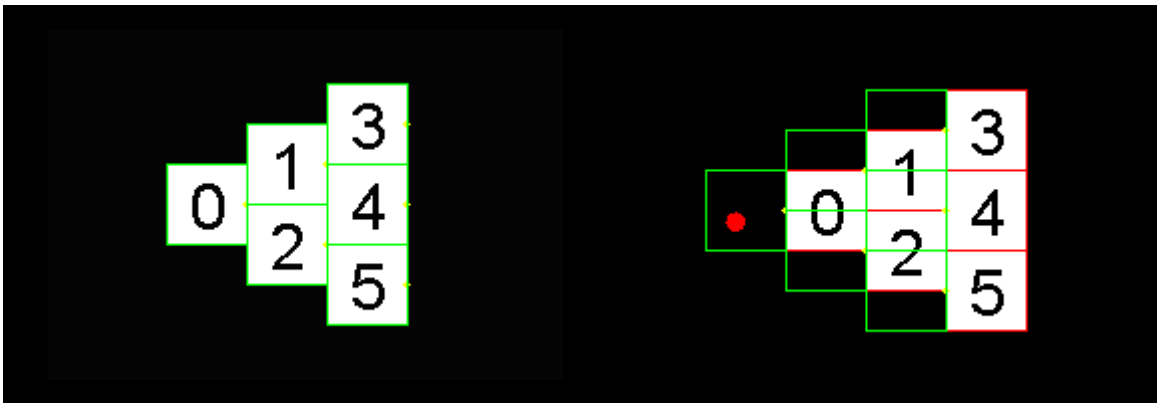

图表 36 测试样例-基本正确性测试-翻转旋转混合

这种情况下，为了避免 6 和 11 都翻转而发生的剧烈缩小，最佳的解决方案是选择翻转之后冲突最少的一个 Label 进行缩小，即 Label-9。

## 4.2 边界测试

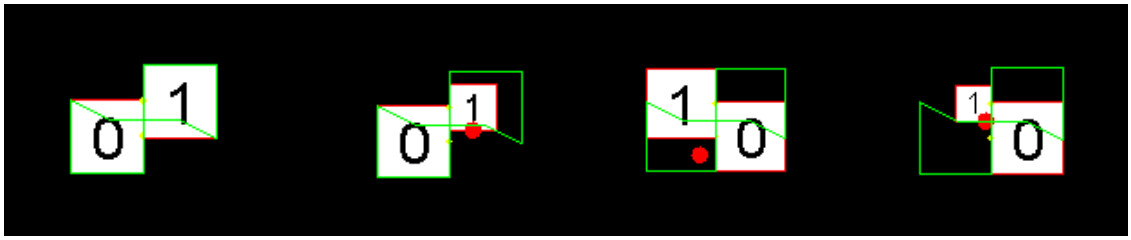
以下几个例子分别测试了 Label 在翻转和缩放的边界情况下结果的正确性。

**紧邻的翻转：**



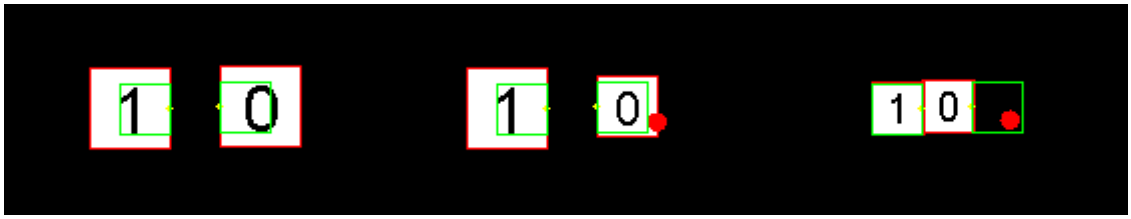

图表 37 测试样例-边界测试-紧邻的翻转

**紧邻的缩小 1**（Label 1 恰好在 Label0 的右上角）：




图表 38 测试样例-边界测试-紧邻的缩小 1

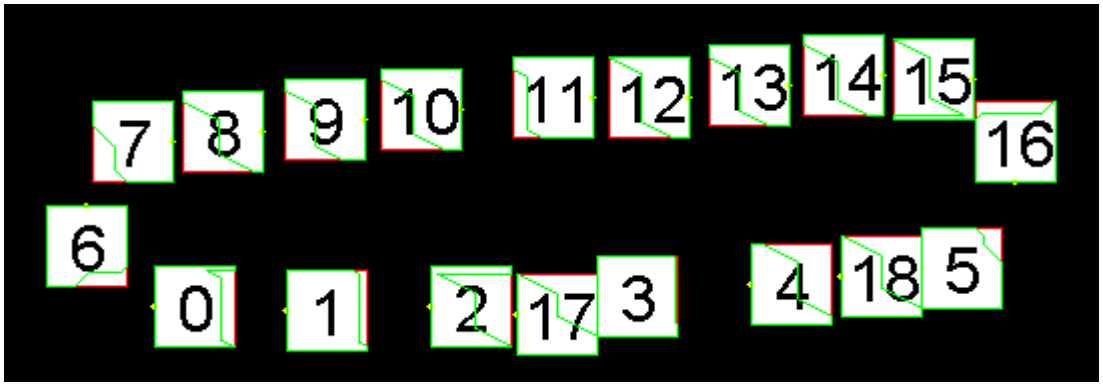
**紧邻的缩小 2**（两个 LabelY 方向差别小于 X 方向差别）：



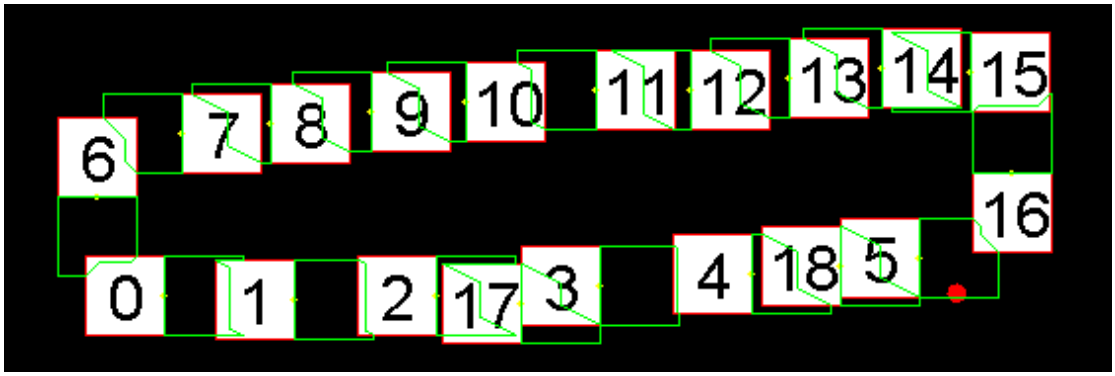

图表 39 测试样例-边界测试-紧邻的缩小 2

在这一测试数据下，我们的程序出现了错误（最右侧图）。在算法描述过程中我们提到了不对称处理的解决方法，但是由于时间所限，我们没有修正这一错误。

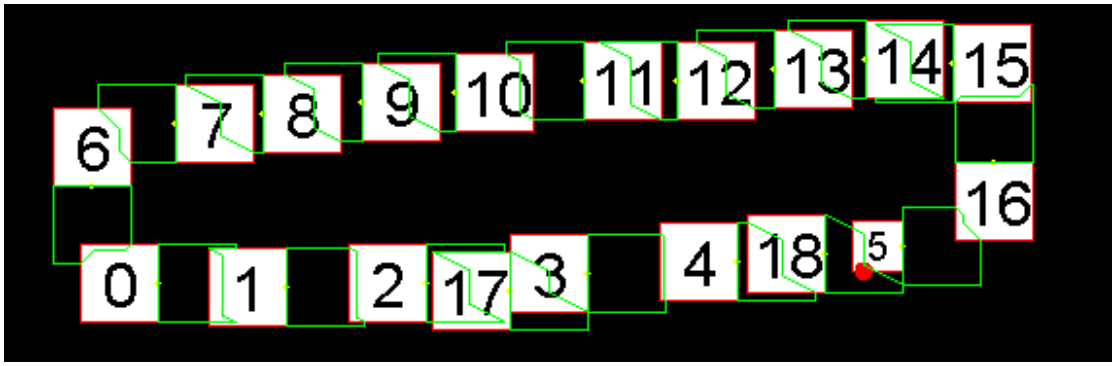
**环状 Label 图和循环覆盖问题：**



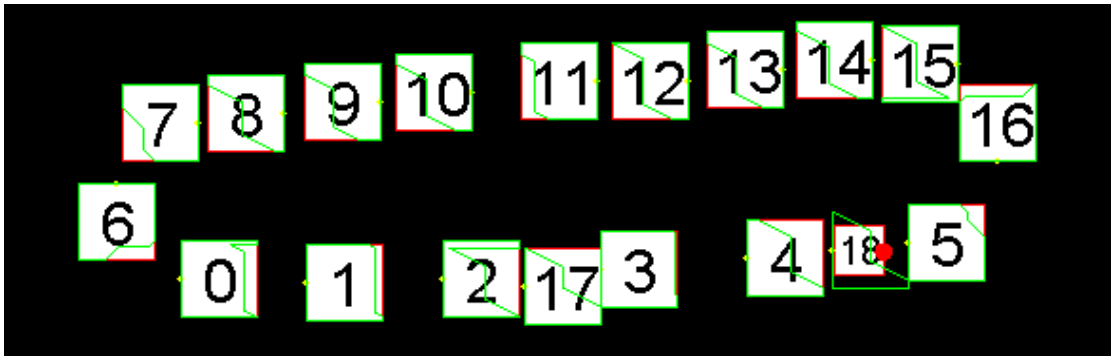

（a）




（b）




（c）




（d）

图表 40 测试样例-边界测试-环状

这组输入数据是成环状的，每一个 Label 与下一个 Label 之间都存在一条 Domino 边。因此最简单的想法就是认为，所有的 Label 都翻转即可（图 b）。但是事实上，在环状翻转回来之后，可能发生再次覆盖的情况，这事实上是多次覆盖的问题，在算法描述部分我们详细描述了处理的方式（图 c，d）。

## 4.3 大数据量测试

最后，我们测试了程序对大数据量的承受能力。

首先我们测试了大数据量的螺旋型和阶梯形数据，包含的 Label 数为 209、420 和 1274，均能够迅速的得出正确的结果（预计算小于 10sec，避让方案计算实时）。

# 5 结论和展望

我们根据[3]中的算法实现程序解决 Point Avoiding 问题，论文中有少量有问题或者没有深入讨论的部分，在我们的实现过程中被一一修改和细化。Point Avoiding 通过定义 Blocking Edge 和 Domino Edge 将整个问题转化到图空间上，求解类似于最大连通子图的权重子图，并将权重子图于最后的方案一一对应。将问题的输入输入分别与图空间对应，这样就能够利用图的一些性质和算法求解问题，这种思想值得我们借鉴。

由于算法本身或者我们实现中的问题，程序还有少量不能正确处理的问题，这在上面的实验结果分析中都已经一一给出，有时间的时候我们会尽快修正。

本问题对于 Label 以及其运动作出了较强的规范，即 Label 必须是正方形，Label 只能绕固定点翻转或者调整大小。实际应用中，如果能够打破这些限制，Label 的外形会更接近实际图形，躲避固定点的调整过程必定会更加平滑，我们主要讨论两种扩展的的难度：

（1）Label 的形状可以扩展为矩形，这样更接近实际应用中的需求。但是这也带来了问题，矩形 Label 的大小不一，如果两个 Label 相交需要调整尺寸的时候就不能采用原先采

用调整大小之后的边长定义这个操作的代价。这主要是由于两个 Label 调整后的大小尺寸可能完全不一样。为了解决这个问题，我们规定矩形 Label 必须按照一定的比例同时调整长宽。两个 Label 相交需要调整大小时，两个 Label 分别需要调整到原先 $\delta_1, \delta_2$，取定 $\min(\delta_1, \delta_2)$ 作为本次调整的代价。这部分代码已经基本完成，由于时间的原因没有放在最终程序中。

（2）可以进一步扩展 Label 运动方式，即允许 Label 绕固定点滑动。在这种情况 Domino Edge 和 Blocking Edge 失去了原有的含义，两个 Label 相互躲闪的时候可能会影响到其他的 Label，原先算法中所描述的冲突图就不能充分描述该问题，这就需要对冲突图进行进行扩展或者摒弃冲突图的做法，算法的复杂度不会低于当前问题（当前问题可以看成是 Label 运动扩展后的一个特例）。

基于原先一些实验和实际工程的经验，我们最后安排了一周半左右的时间测试程序、修改 bug，这让我们有充足的时间发现和解决算法中出现的各种问题。同样的在这个过程中，我们对算法的效率有了非常直观的认识，这在上面的试验结果和分析中都有详细的讨论。

# 6 分工和进度

实验分工如下

| 学号 | 姓名 | 任务 |
|---|---|---|
| 2004211033 | 张力 | 界面设计、构造测试样例 |
| 2004310411 | 孙洛 | 基本结构、计算 Flipping Region |
| 2004310436 | 徐俊 | 构建冲突图、计算权重子图 |

实验进度如下

| 周次 | 进度 |
|---|---|
| 11.22-11.28 | 准备开发环境，完成基本数据结构 |
| 11.29-12.05 | 构建冲突图、计算权重子图、计算冲突图、界面设计同步进行 |
| 12.06-12.12 | 界面和算法部分联调、完成第一个 Demo 版本 |
| 12.13-12.19 | 大规模测试、不断修改错误 |

# 7 参考文献

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

[2] Doddi, S., Marathe, M.V., Mirzaian, A., Moret, B.M., Zhu, B.: Map labeling and its generalizations. In: Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms (SODA'97), New Orleans, LA (1997) 148–157

[3]Farshad, R., Mohammad, G.: A Fast Algorithm for Updating a Labeling to Avoid a Moving Point. CCCG'04
[4] M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf Computational Geometry Algorithms and Applications second edition
[5]Point Location Applet, http://www.students.tut.fi/~kaartine/comp/applet.html
[6]计算几何讲义，邓俊辉