计算几何大作业
最大宽度矩形、正方形空环求解

王宣润 2020215219
李哲 2019310957
李玮祺 2020215216

2021 年 6 月 18 日

1 问题描述

计算点集中最小/最大尺寸几何对象的问题是计算几何的重要研究内容之一，在这项工作中我们希望解决的问题是，对于二维平面上的若干点，找出一个**边界平行于坐标轴**的正方形或矩形环带，使其中不含有任何点，并最大化它的宽度。

在本文中，我们考虑的是二维平面直角坐标系中的问题。我们定义一个正方形的中心为该正方形的两条对角线交点，正方形半径为边长的一半。正方形空环带的定义是，两个四边与坐标轴平行且同中心的正方形边界之间的部分，环形的宽度 w 如图1（左）所示，是两个正方形对应方向边界之间的距离。矩形空环带的定义是，从一个与坐标轴平行的矩形 R 内部减去另一个与坐标轴平行的矩形 R' 而得的区域。如图1（右）所示，两个矩形对应边之间距离为 t, r, l, b，我们定义两个矩形之间环带的宽度为这四个距离中的最小值。与正方形环带的定义不同，矩形环带不要求两个矩形有相同的中心（对角线交点）。

为了让问题有意义，要求内矩形/正方形的边界或者内部、外矩形/正方形的边界或者外部至少要有一个点。另外，问题允许矩形/正方形退化的情况，即外侧的矩形/正方形允许有某几条边处于无穷远处，这时矩形/正方形环带会退化成直线条带或者 L 形条带。但不允许外矩形/正方形所有边退化至无穷远（这时不符合外矩形/正方形边界或外部至少包含一个点的约束），但允许内部矩形/正方形退化到只剩一个点。

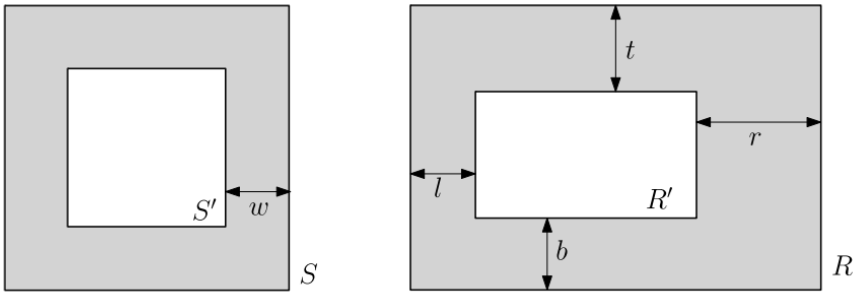

图 1: 正方形空环带和矩形空环带

2 算法说明

2.1 符号定义

符号	意义
P	平面上的点集
n	$\|P\|$，即点集大小
p_i	平面点集中的第 i 个点
$x(p)$	点 p 的 x 坐标
$y(p)$	点 p 的 y 坐标

表 1: 符号表

我们将正方形空环带和矩形空环带的情况分开处理。

2.2 正方形空环带

首先可以证明 (证明略，见 [2])，正方形空环带一定属于以下三种情况之一：

1. 外正方形的两条对边上各有一个点
2. 外正方形的两条邻边上各有一个点（L 形区域）
3. 外正方形的一条边上有一个点，其他三条边都是无穷（条带）

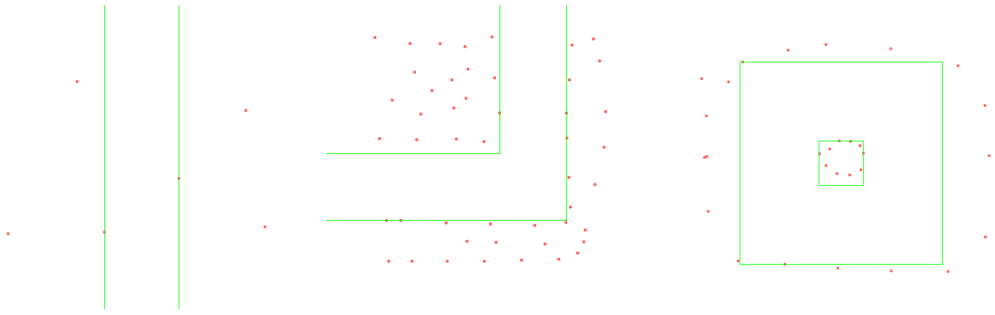

图 2: 三种情况的示例

情况 3，空环带退化成了空条带，它的求解方式很简单，可以直接按照 x 坐标或者 y 坐标排序后扫描一遍，查找其最大的 gap，即为空条带的宽度。总时间复杂度 $O(nlogn)$。

情况 2，以每个点的 x 坐标和 y 坐标画垂直线和水平线，这些线之间有 n^2 个交点，对于每个这样的交点 o，找到以它为原点的第一象限里 (不含坐标轴)，x 坐标最小的点和 y 坐标最小的点，以此为限制找到 L 形区域，时间复杂度 $O(n^2logn)$[4]。

情况 1 较为复杂，先对所有点按 y 坐标排序，枚举外正方形顶边上的那个点 p_i 和底边上的那个点 p_j，然后以 $y = \frac{y(p_i)+y(p_j)}{2}$ 画线，这条线即是中线，正方形空环带的中心 c 一定在这条中线的一条线段上（因为需要顶边和底边都有 1 个点），且这条线段的左右端点会被 p_i 和 p_j 确定。具体而言，这条线段的左右端点

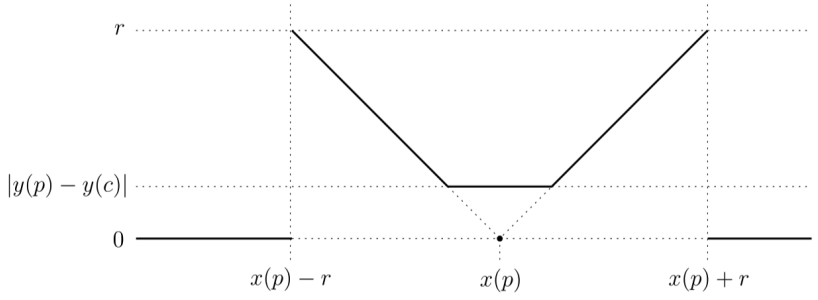

图 3: $f_p(c)$ 的函数图像

的横坐标 $x \in [min\{x(p_i), x(p_j)\} - r, max\{x(p_i), x(p_j)\} + r]$。设外正方形的边长为 $2r$，定义函数 $f_p(c)$ 表示当正方形空环带中心点位于位置 c 时，p 点对环带宽度的限制，即 $f_p(c) = ||p - c||_\infty$。

假设环带中心点 c 在中心线上的 $x = c$ 这个位置上，那么要使得 p 这个点不在空环带中，则环带宽度不能大于 $r - ||p - c||_\infty$，那么 f 的图像应如图3所示，图中中间平的一段的物理意义是此时 y 方向距离比 x 方向距离大，旁边为 0 的区域表示如果 p 落在了外正方形外，则其对结果没有限制。

那么，我们把所有 $p \in P$ 的 $f_p(c)$ 都求出来，然后求一个包络线，即

$$f(c) = \max_{p \in P} f_p(c)$$

求 $f(c)$ 的最小值点，即可知道 c 为何值时 $w = r - f(c)$ 最大。

求上述包络线的过程，我们可以发现上述每个点的图像都可以拆成三个部分，即斜率为 $1, 0, -1$ 的三个部分。将三个部分分开处理，再合并起来就能得到总体的包络线了。另外，由于一开始就把所有的点按照 x 和 y 分别排过序了，所以这里处理包络线时也不需要进行排序。包络线只会有 $O(n)$ 段，故这里处理包络线的过程是 $O(n)$ 的。注意到最优解一定会出现在包络线的转折点，或者在包络线之间的交点处，这是因为如果最优解不在上述的位置，一定可以通过在包络线上平移，而得到一个不更差的解，且其在转折点或交点处。所以我们只需要对所有转折点和交点检查可行性即可。加上之前枚举 p_i 和 p_j 的时间消耗，情况 1 处理的总时间复杂度是 $O(n^3)$ 的。为了 coding 的方便，在实际代码编写过程中，我将上述的定义反了过来，即 $f'_p(c) = r - ||p - c||_\infty$，$f'_p(c)$ 表示 p 点限制条带宽度 $w \le f'_p(c)$。

由于这三种情况可以囊括所有的可能情况，所以求解正方形空环带算法的整体时间复杂度是 $O(n^3)$。

将上面的查询步骤形式化，我们可以得到如下的对数据结构的操作要求：

1. 对于每个点，查询这个点右上方区域（其他方向的区域可以将坐标系旋转）中，x 坐标最小的点和 y 坐标最小的点。要求单次查询时间复杂度 $O(logn)$。

2. 维护三条包络线，每条包络线的元素分别全是斜率为 $1, 0, -1$ 的线段。其中斜率为 $1, -1$ 的线段的 x 坐标跨度为 r，斜率为 0 的线段的 x 坐标跨度为 $2r$。包络线数据结构需要回答它们在每个 x 处的最低点，并且这三条包络线需要两两求交，时间复杂度要求 $O(n)$。此处的分析可见3.2一节。

利用数据结构解决了上述子问题之后，本问题的总时间复杂度是 $O(n^3)$，和原论文 [2] 的描述一致。

2.3 矩形空环带

矩形的情况略为复杂，首先矩形空环带也可以分成上述的三种情况，两种退化情况的处理方式同正方形空环带，故下只考虑一般情况。因为每个方向的宽度都不一样。首先可以证明

- 内矩形的每条边上要么至少有一个点，要么是无穷
- 外矩形的每条边上要么至少有一个点，要么是无穷

这是因为如果外矩形的某条边上没有点，那可以向外平移这条边，从而使其碰到一个点或者到无穷远，而此时该矩形空环带是不会更窄的；如果内矩形的某条边上没有点，那也可以向内平移这条边，从而碰到一个点，此时矩形空环带也不会变窄。

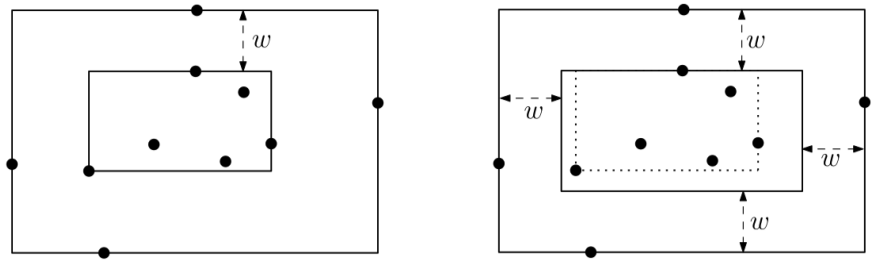

图 4: 扩展矩形空环带，使其四边的宽度相等

由于矩形空环带的宽度由其最窄的部分确定，所以那些宽的方向可以扩展，使得其每个方向的宽都是 w。如图4，若扩展前矩形顶边的宽度最小且为 w，此时该环带的宽度由内外矩形的顶边确定。不失一般性可以只考虑这一种情况。

接着我们可以考虑枚举外矩形顶边的点 t 和底边的点 b，继续寻找空环带的最大宽度 w。注意到 w 满足单调性：如果给定一对 t 和 b，且已知 $w_2 > w_1$ 且 w_2 是可行的，则 w_1 也是可行的。

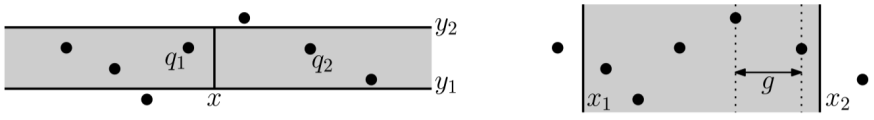

图 5: 判断 w 值是否可行的两个步骤图示

怎样判决一个 w 值是可行的呢？分为以下两个步骤：

1. 判断顶边下方和底边上方的 w 宽度内的合法点范围，x 是上述确定的 $t(b)$ 点的横坐标，$|y_1 - y_2| = w$，如左图可以找到竖直线段 $(x, y_1) - (x, y_2)$ 水平往左平移遇到的第一个点 $q_{1t}(q_{1b})$ 和往右平移遇到的第一个点 $q_{2t}(q_{2b})$。由于我们需要确保 $t(b)$ 点是在外矩形上，所以 $t, b, q_{1t}, q_{2t}, q_{1b}, q_{2b}$ 这几个点需要满足合适的位置关系，即 $q_{XX}(q_{XX} : q_{1t}, q_{1b}, q_{2t}, q_{2b})$ 不能出现在 $x(t)$ 到 $x(b)$ 之间，因为一旦出现在在 $x(t)$ 到 $x(b)$ 之间，说明顶边下方和底边上方宽为 w 的条带内有点存在，说明宽为 w 是不合法的；且 q_{XX} 实际上卡住了这个矩形，因为如果空环带宽为 w，这个条带的范围里是不能有点的，为此这个环带的外矩形最多延伸到 q_{XX} 处。

2. 确定好外矩形范围之后需要判断内矩形的合法范围。以 q_{XX} 和 t、b 的 x 坐标作为边界参考量，寻找 $y(p_i)$ 满足 $y(p_i) \in [y(b), y(t)]$ 和 $x(p_i) \in [min\{q_{1t}, q_{1b}\}, max\{q_{2t}, q_{2b}\}]$，查询所有相邻 p_i 点的 gap，是否有宽度大于等于 w 的两个 gap。这实际上是在找能放下矩形空环带两个侧边的位置，因为侧边内不能有点，所以如果有宽度大于等于 w 的 gap，就可以把侧边放置在这个 gap 里。如果查询到左右两边

的两个 gap 值都大于等于 w 且两个 gap 之间有至少 1 个点（内矩形中至少要有 1 个点），说明宽度为 w 是可行的。这是因为可以把矩形空环带的左右两边卡在这两个 gap 里，而这两个 gap 里在 $y(t)$ 和 $y(b)$ 之间是一定没有任何一个点的。注意这里是要有 2 个 gap，只有 1 个宽度大于等于 w 的 gap 是不可行的，因为这样内矩形中就没有任何点了，不符合我们的要求。

从总的算法出发，枚举顶边点 t，再枚举底边点 b，还需枚举 w，但注意到在定下 t 之后，w 只需从当前历史最大的 w 枚举，并且由于前面所说的单调性，一旦某个 w 不可行就可以停止枚举，因为给定 t, b 后，如果 w_0 不合法，那么 $w_1(w_1 > w_0)$ 更不可能合法。并且由于内外两个矩形的边上都至少存在 1 个点，所以 w 的取值只会有 $O(n)$ 种。这样实际上如果将 (w, b) 看成是一个坐标，整个枚举过程实际上是从 $(0, 0)$ 到 (n, n) 的一条只能往右或往上走的路径，这条路径的长度是 $O(n)$ 的，所以总的 (t, b, w) 三元组的个数是 $O(n^2)$ 的。对于每个三元组，都会用上述的数据结构来查询和插入一次相关的信息，所以总时间复杂度是 $O(n^2 logn)$ 的，忠实还原了原文 [2] 中的时间复杂度。

将上面的查询步骤形式化，我们可以得到如下的对数据结构的操作要求：

1. 对于每个点 p_i，找到线段 $(x(p_i), y(p_i)) - (x(p_i), y(p_i) + w)$ 以及线段 $(x(p_i), y(p_i)) - (x(p_i), y(p_i) - w)$ 往左，往右水平平移碰到的第一个点。要求单次查询时间复杂度 $O(logn)$。

2. 对于某区域 $x \in [x_i, x_j], y \in [y_m, y_n]$，查询最左侧的宽度大于等于 w 的 gap 和最右侧的宽度大于等于 w 的 gap。要求单次查询时间复杂度 $O(logn)$。

3. 对于某区域 $x \in [x_i, x_j], y \in [y_m, y_n]$，查询这个区域内是否存在点。要求单次查询时间复杂度 $O(logn)$。

3 数据结构和算法设计

3.1 求解右上方区域的极值点

由于我们需要对每个点求右上方区域的极值点，所以可以通过合理调整求解顺序来降低复杂度。首先将所有点按照 y 坐标排序，然后按 y 坐标从大到小的顺序将每个点的 x 坐标插入平衡树。在插入之前先在平衡树中查询第一个 x 坐标大于其本身 x 坐标的点（即 upperbound 操作）。由于点是按照 y 从大到小排序的，所以每次查询实际都保证了只会查到 y 坐标更大的点。而每次的查询操作都满足了查到的一定是 x 坐标恰好更大的那个点，即该点右上方区域中 x 坐标最小的点。同理将 x, y 镜像，得到的就是该点右上方区域中 y 坐标最小的点。

上述操作对于每个点都会操作 1 次平衡树查询和 1 次平衡树插入，加上一开始的排序 $O(nlogn)$，所以总时间复杂度是 $O(nlogn)$ 的。

3.2 维护包络线

首先我们来回顾一下前述的需求：在枚举完外正方形顶边的点 t 和底边的点 b 后，这两个点中间所夹的所有点都会对正方形空环带的宽度有所限制。其限制具体到函数图像而言就是一个等腰梯形，见图6，其两个腰交于正中央。观察前述的函数目标，我们需要选择一个 (x_0, y_0) 使得其在所有的包含 x_0 的蓝色梯形边的下方且此时 y_0 最大。为了叙述和编码方便，在蓝色梯形左右侧，函数的值为 0，此时该蓝色梯形不对 y 有任何限制。可以发现一**个点在蓝色梯形内**等价于这个**点同时在这个蓝色梯形三条边延伸的范围**之下。所以我们可以将三种边分开考虑，分别对其求包络线，再汇总起来即可。

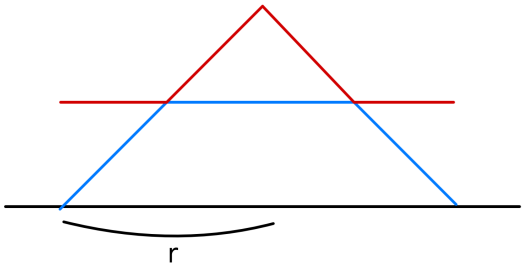

图 6: 三段线段图示。其中蓝色的等腰梯形是一个点对正方形空环带宽度的限制，而红色的线是蓝色梯形三条边的延伸。**一个点在蓝色梯形内**等价于这个**点同时在三条边延伸的范围**之下。由函数图像的性质可以知道，斜率为 1 的左侧腰和斜率为 -1 的右侧腰相交于正中央，且这两条腰加上延长部分在 x 坐标上的跨度为 r。梯形顶边加上延长部分在 x 坐标上的跨度为 $2r$。

求斜率为 1 边的线段包络线非常简单，只需将所有点按照 x 坐标排序，然后依次插入并维护上一个点和当前点的位置即可。因为每个点延伸出来的线段的斜率都是 1，是互相平行的，所以不用考虑这些线段之间会有交点；因为我们的目的是每个候选结果需要在这些线段的下方，所以包络线要求的是这些线段在每个 x 处的最小值。斜率为 -1 的线段包络线求法同理，将 x 坐标取反即可。该步骤时间复杂度 $O(n)$。

斜率为 0 的线段的包络线就略为复杂。因为它会涉及到更低、更高线段互相遮盖的问题。这里需要使用一个单调双端队列来处理，首先将所有的点按 x 坐标排序，然后从左到右单调队列维护**将来可能会对包络线产生贡献的线段**：如果某两条线段 a, b，它的 y 坐标满足 $y(a) > y(b)$，右端点 x 坐标满足 $x(a) < x(b)$，此时 a 线段将永远不可能对包络线产生贡献，因为随着 x 坐标的增大，a 线段将首先结束，而其全程被 b 线段在其下方遮盖，在从当前 x 开始到将来都不会有 a 的任何一段成为结果。所以队列中维护的线段将保持 x 和 y 的递增。由于每条线段都只进队出队 1 次，所以这一步的时间复杂度是 $O(n)$，另外由于可以从已经有序的排序结果里直接获取 x 有序的点，所以此处求包络线部分的排序实际上是一个 y 坐标的筛选（在 $[y(b), y(t)]$ 之中）过程，是 $O(n)$ 的。

在求解完三条包络线之后，再将它们两两求交点。根据前述，最优解只会存在在包络线交点或者转折点处，所以求交是必不可少的。依然可以通过线性扫描的方法，类似于多个有序序列合并问题的思路，将每个交点求出。最后再将每个交点和每个转折点分别判断是否同时在三条包络线的下方，这一步的思路和上面求交点的思路非常类似。这些特殊情况处理非常考验细节，涉及到竖直线段和线段端点的处理，可以参考代码中的实现，这里就不赘述了。这一步的时间复杂度也是 $O(n)$ 的。

3.3 求解线段平移相遇首点

对于此问题，我们的方法并没有参考原论文中使用的 Segment Dragging 技术 [3]，而是另辟蹊径采用了一个离线算法。为了介绍的简单起见，我们只考虑求每个点上方线段向右平移遇到的第一个点，其他三个方向可以对照同理进行计算。

我们需要对每个点 (x_i, y_i) 求解**由它出发向上的一条线段** $(x_i, y_i) - (x_i + w, y_i)$ **向右平移遇到的第一个点**，这个性质非常关键，因为每一条线段都是从这个点出发的，而所有的点只有 n 个，故可以针对每个点进行预处理。注意到可能构成结果的点其实非常有限：首先其他三个象限里的点都不可能成为结果，然后在第一象限里，只有在阶梯上的点才会成为结果，见图7。所以我们可以对每个点都求一个阶梯，然后查询的时候在阶梯上二分查找即可。这样预处理的时间复杂度是 $O(n^2)$，空间复杂度也是 $O(n^2)$。每次查询的时间复杂

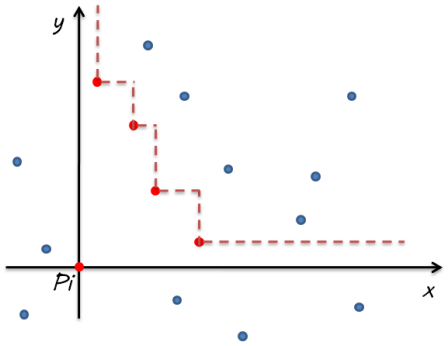

图 7: 阶梯图示，对于 p_i，只有红色的点才有可能成为结果，蓝色的点要么在其他三个象限，要么被红色的点构成的阶梯完全覆盖，故不可能成为结果。

度是 $O(logn)$。

3.4 求解最左侧最右侧宽度大于等于 w 的 gap

本问题其实可以用 Range Tree 解决，Range Tree 的每个节点都保存当前区间内的最大 gap 和最左最右点。但 Range Tree 有个问题，它只支持坐标为整数的情况，考虑到可扩展性，我们使用了 Splay 平衡树作为它的替代品。

Splay 树的相关操作和复杂度分析在这里也不再展开叙述；原问题是一个离线二维问题，但实际上可以通过巧妙设计求解顺序来将 y 维度去掉，所以变为了在线一维问题。我们知道，Splay 平衡树支持提取区间的操作，对于区间 $[l,r]$，只需获取 l 的前驱节点 l' 和 r 的后继节点 r'，然后将 l' 旋至树根，将 r' 旋至 l' 的右孩子，则树根的右孩子的左孩子这棵子树即代表了区间 $[l,r]$。对于 Splay 的每个节点，维护其子树的最左点、最右点和最大 gap。每次旋转的时候，都需要根据新的节点关系来重新维护这三个信息。查询的时候只需要关注各个节点子树的最大 gap 值和左、右子树的最右、最左值，以确定应该继续往哪棵子树下探查询。插入一个新点的操作也很简单，和普通的平衡树插入相同，只需注意维护每个节点的信息即可。

由于使用的是 Splay 平衡树，其每次操作的期望时间复杂度是 $O(logn)$，空间复杂度是 $O(n)$。

3.5 求解区域中是否存在点

有了上面的 Splay 平衡树，可以直接将原问题降维后，提取左右两个端点的区间，然后判断该区间对应的子树是否为空。时间复杂度 $O(logn)$。

4 鲁棒性与精度

我们一开始为了避免精度问题，就将所有的点坐标均设置为整数。但考虑到不同的需求和任务，我们所有的算法都支持浮点数运算，只需要改动少量的比较部分即可完善对浮点数坐标的支持。我们的程序做了较为完善的错误处理和输入检查，鲁棒性很高，正常使用的情况下基本不会出现运行时错误等问题。

5 吞吐能力和性能

为了充分测试我们程序的性能，采用随机生成点的方法，记录点数和用时的关系可以获得下表。

点数	正方形空环带 ($O(n^3logn)$)	矩形空环带 ($O(n^2logn)$)
30	70	9
50	282	24
100	1934	90
200	13932	355
500	195510	1949
1000	-	6771
2000	-	21768

表 2: 性能测试（单位：毫秒）

注：由于我们编写代码时在求解正方形空环带的算法中无意加入了对交点和包络线转折点进行排序的操作，在撰写报告时才发现这个隐藏的问题，这里求解正方形空环带的算法的时间复杂度实际是 $O(n^3logn)$ 的。要解决这个问题可以参考多个有序序列合并的算法，但再次对此进行修改的代价较大，所以就将此问题遗留了下来。从上方的用时表里也可以发现实际上求解正方形空环带的用时比求解矩形空环带大了一个 n 的量级。

6 小组分工

王宣润 算法实现、报告算法部分撰写

李哲 UI 实现、用户手册撰写、报告视频的录制

李玮祺 文献综述和背景介绍、生成数据

7 实验总结

在这项工作中，我们解决了1问题描述中提出的平面点集上求解最大矩形/正方形环带的问题。其中求解正方形空环带的整体时间复杂度是 $O(n^3)$，空间复杂度是 $O(n)$；求解矩形空环带的整体时间复杂度是 $O(n^2logn)$，空间复杂度是 $O(n^2)$。经过测试，算法能在预期时间和空间下完成问题的正确求解，在较大的数据规模下也有良好表现。我们还设计了可交互的 UI 界面以提高算法的展示效果。

8 相关工作

计算点集中最小/最大尺寸几何对象的问题是计算几何的重要研究内容之一，该类问题已经被广泛研究并应用于不同几何形状上，包括圆形、矩形、环形等。求最大空圆可以利用 Voronoi 图交点的性质在 $O(nlogn)$ 时间内完成 [8]。在工作 [1] 中，作者研究了如何更好计算与坐标轴平行的最大面积的矩形，提出了在 $O(nlog^2n)$ 时间内运行的算法。目前本项目关注的最大空环问题是一个比较新的研究领域。之前的研

究 [5] 研究了最大空圆环的问题，作者通过枚举点集中的点对，计算点对中以该点为顶点的抛物线交点得到圆心半径，给出了时间复杂度为 $O(n^3logn)$，空间复杂度为 $O(n)$ 的算法。项目中最大矩形环带的定义来源于工作 [7]，但在该工作中矩形的四边可以不平行于坐标轴，在该工作作者提出了基于旋转卡壳和 Davenport–Schinzel 序列的方法，最终实现了时间复杂度 $O(n^2logn)$，空间复杂度 $O(n)$ 的算法。本次大作业的参考论文 [2] 在此基础上进一步进行研究，针对四边平行于坐标轴的情况，提出了对于正方形空环带和矩形空环带时间复杂度分别为 $O(n^3)$ 和 $O(n^2logn)$ 的解法。对于正方形空环和矩形空环的退化形式，工作 [6] 研究了在点集中寻找最宽空带的情况，工作 [4] 考虑了 L 形拐角的问题，作者提出了时空复杂度为 $O(n^3)$ 的解法。

9 应用

求解最大宽度空环带问题可以用于以下的场景：

1. 需要开挖一条公路，把地图上散落分布的某些村镇围起来，并限制公路只能是东西和南北走向。问公路最大宽度为何，并需要给出挖掘方案。
2. 因为火情影响，地图上有若干个着火点，需要留出足够的宽度空间用来布置防火带，以阻止着火点进一步合并成更大的着火区域。这些留空区域只能沿着南北和东西走向。问防火带最大宽度为何，该如何布置?
3. .....

参考文献

[1] Alok Aggarwal and Subhash Suri. Fast algorithms for computing the largest empty rectangle. In *Proceedings of the third annual symposium on Computational geometry*, pages 278–290, 1987.

[2] Sang Won Bae, Arpita Baral, and Priya Ranjan Sinha Mahapatra. Maximum-width empty square and rectangular annulus. *Computational Geometry*, 96:101747, 2021.

[3] Bernard Chazelle. An algorithm for segment-dragging and its implementation. *Algorithmica*, 3(1):205–221, 1988.

[4] Siu-Wing Cheng. Widest empty l-shaped corridor. *Information Processing Letters*, 58(6):277–283, 1996.

[5] José Miguel Díaz-Báñez, Ferran Hurtado, Henk Meijer, David Rappaport, and Joan Antoni Sellarès. The largest empty annulus problem. *International Journal of Computational Geometry & Applications*, 13(04):317–325, 2003.

[6] M Houle and A Maciel. Finding the widest empty corridor through a set of points. *Snapshots of computational and discrete geometry*, pages 210–213, 1988.

[7] Joydeep Mukherjee, Priya Ranjan Sinha Mahapatra, Arindam Karmakar, and Sandip Das. Minimum-width rectangular annulus. *Theoretical Computer Science*, 508:74–80, 2013.

[8] Godfried T Toussaint. Computing largest empty circles with location constraints. *International journal of computer & information sciences*, 12(5):347–358, 1983.