正交多边形中灯塔问题的相关算法实现

琚锡廷 (2020310789)
刘应天 (2020210959)
胡正丁 (2019211111)

目录

1 问题描述

灯塔问题是著名的美术馆问题的一个变种。该问题场景描述如下：给定一个航行海域和一个航行目标，要求给出所需灯塔的最小个数以及它们的位置和编号，使得从这个区域内任意一点出发的船可以根据给定的导航方式到达终点。这里的导航方式指的是每次向可见的具有最小编号的灯塔移动。

以上的描述中，目标、区域、灯塔、船只，以及灯塔的可见性的定义都是可变的，所以这些定义的不同会引出各种灯塔问题。本文所提的灯塔问题指[3] 所讨论的灯塔问题的一种，条带状灯塔问题 (strip lighthouse problem，下称 SLP 问题)：航行区域是一个二维平面上的正交多边形 P，正交的性质将该问题从 NP 问题简化为多项式时间求解的问题；船只是多边形内的一个点 q；灯塔均指条带状的灯塔 (strip lighthouses)，即位于多边形 P 内，满足正交性质且与 P 有两个交点的一条线段 l；l 对点 q 可见的定义为 q 在 l 上的垂直投影在多边形 P 内部；目标 t 是区域上的一条边，同时也一定是最终结果里编号最小的灯塔。

如图 1所示，正交多边形中的红色线段代表终点，绿色线段和终点构成 SLP 解的集合。多边形中的每一个点都可以通过灯塔导航前往终点。

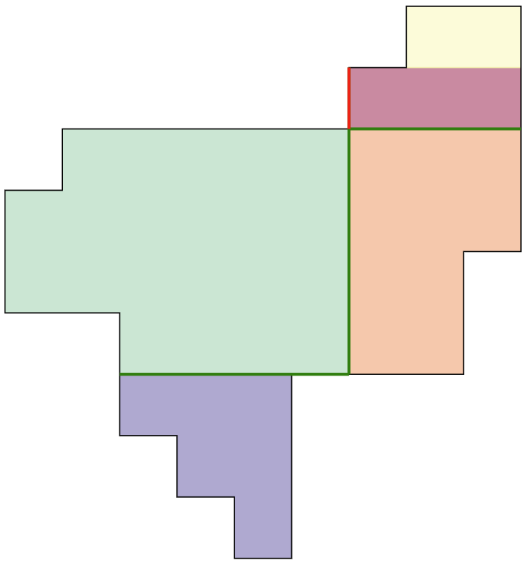

图 1: 灯塔问题定义及其解

2 算法调研

我们的调研的算法主要来自 [3] 提出的平方级多项式时间算法。值得一提的是，该文作者在投稿文章后发现了一个文中定理的反例证明，因此我们对算法正确性进行了进一步探究，并在第 3 章中对算法描述和设计作了一定的优化和更改。我们认为，我们实现的算法可以应用于绝大部分通用情况，并取得良好的运行效率。

2.1 测例生成

为了提供丰富的测例，我们需要实现一个随机正交多边形生成算法，即给定正交多边形的顶点数，生成随机的正交多边形，这一部分我们实现了 [5] 中的算法。[5] 中主要提出了两种生成正交多边形的方法，第一种是一个多项式时间算法，可以在 $O(n^2)$ 时间、$O(n)$ 空间内给出一个具有 n 个顶点的正交多边形，这个算法从一个正方形开始，以一种迭代的方法逐渐增加顶点数直到具有 n 个顶点。第二种是一个约束编程算法，给定正交多边形顶点满足的条件，在约束空间内搜索，这种方法可以生成所有的具有 n 个顶

点的正交多边形，但是速度较慢。对于后一种方法，根据 [5] 中的实现，作者只尝试了 $n < 90$ 的情况，因此我们认为并不适合用来作为此问题中测例的生成。因此以下主要介绍 [5] 中描述的第一个算法。

规定一个正交多边形 P 如下描述：P 具有 n 个顶点，$v_1, v_2, ..., v_n$ 为 P 的顶点，以逆时针方向给出，则 P 由 $v_1, v_2, ..., v_n$ 唯一定义。定义 P 的边 $e_i = \overline{v_i v_{i+1}}$(模 n 意义下的)，对于每一条边，P 都将局部地在其左侧。每一个并不在一般位置的正交多边形 (指其存在两条边共直线) 都可以通过一个 ϵ 扰动使其任意两条边都不共直线，因此该算法主要将生成限制在一般位置的正交多边形。由于共线的情况并不会影响灯塔问题的结果，因此我们直接采用了这个生成算法。值得一提的是，存在边共线的正交多边形可以通过简单地平移一般位置的正交多边形的边来产生。

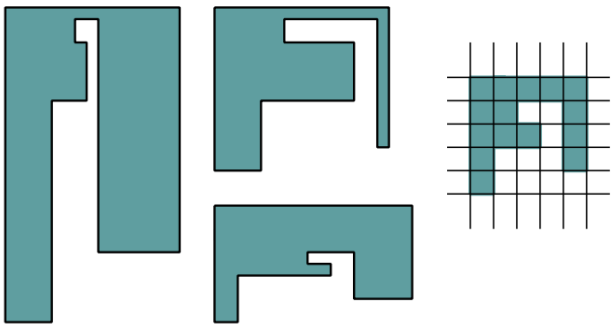

图 2: 用一个格点正交多边形代表左侧的三个一般正交多边形，$n = 12$

首先 [5] 中规定了格点正交多边形的定义，对于一个一般的正交多边形 P，其顶点数为 n，其对应的格点正交多边形为：将 P 的所有水平边按从上到下排序，所有垂直边按从左到右排序，而后我们给每一条边它的排序序号 (从 1 开始)，如果 $\mathcal{V}_{i-1}, \mathcal{H}_i$ 分别是赋给垂直边 e_{i-1} 和水平边 e_i 的的序号，那么 $(\mathcal{V}_{i-1}, \mathcal{H}_i)$ 将是顶点 v_i 的坐标 (如图 2)。因此在这样的定义下格点的左上角坐标为 $(1, 1)$,x 正方向为水平向右，y 的正方向为竖直向下。每一个格点形都将代表一系列的正交多边形，给定一个格点形，我们可以通过在保证格线的相对位置的情况下随机格线间的距离来生成该系列中的正交多边形。而在灯塔问题中，由于可见性与距离无关，因此实际上是否生成一般的正交多边形并无影响，所以在后续的测试中，我们直接使用了格点正交多边形作为输入。以下称一个具有 n 个顶点的正交多边形为 n-正交多边形。

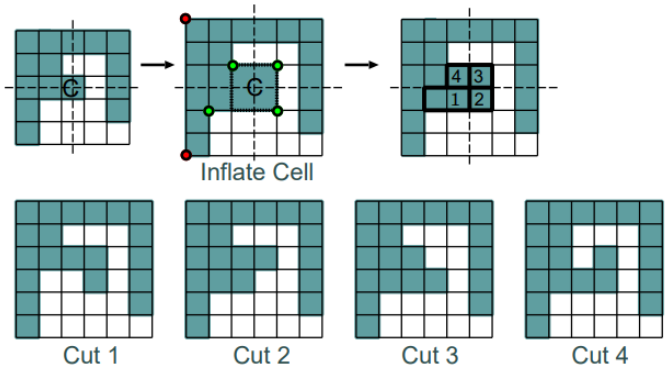

图 3: Inflate-Cut 过程，从 12-正交多边形生成 14-正交多边形。

格点正交多边形生成算法的主要思路是从一个最初的正交多边形 (即正方形) 开始，逐步增加顶点数，并且维护正交性质，直到顶点数达到 n。图3给出了一个从 n-正交多边形生成 $(n+2)$-正交多边形的示例，[5] 将这个过程称为 Inflate-Cut 过程。通过将格子 C 膨胀，并且以其中心点作为切割下来的矩形的顶点，可以生成 Cut 1 4 四种可能的 14-正交多边形。设 $v_i = (x_i.y_i), i = 1, 2, ..., n$ 为 P 的顶点，C 是位于 P 内部的一个单元格 $(1 \times 1$ 大小)，其左上顶点的坐标为 (p, q)。

Inflate 过程：以单元格 C 对 P 进行 Inflate 操作的结果是一个新的具有 n 个顶点的正交多边形 $\tilde{P}$，它的顶点为 $\tilde{v}_i = (\tilde{x}_i, \tilde{y}_i)$，其中

$$\tilde{x_i} = \begin{cases} x_i & , x_i \leq p \\ x_i + 1 & , x_i > p \end{cases}, \tilde{y_i} = \begin{cases} y_i & , y_i \leq q \\ y_i + 1 & , y_i > q \end{cases}$$

在 $\tilde{P}$ 中，直线 $x = p + 1, y = q + 1$ 上将不存在边，它们相交于膨胀后的 C 的中心点，这里将产生新的顶点，并进行 Cut 操作。

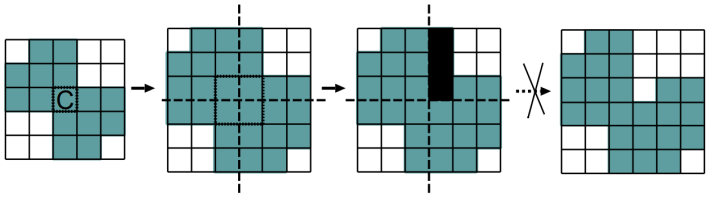

图 4: Cut 过程失败，因为尝试切割的矩形包括了 $\tilde{P}$ 中的两个或以上顶点。

Cut 过程：令 $\tilde{v_C} = (p+1, q+1)$，计算直线 $x = p+1, y = q+1$ 和 $\tilde{P}$ 的边界的四个交点，即四条从 $\tilde{v_C}$ 开始的水平及垂直的射线和 $\tilde{P}$ 边界的第一个交点。随机地选择其中一个交点，称之为 $\tilde{s}$，令 $\tilde{v_m}$ 是 $\tilde{s}$ 所在的边的其中一个顶点。如果由 $\tilde{v_C}, \tilde{v_m}$ 定义的矩形中不包含 $\tilde{P}$ 中除了 $\tilde{v_m}$ 的其他顶点，那么我们就可以切下这个矩形。如果这样的矩形是可切的，令 $\tilde{s'} = \tilde{v_C} + (\tilde{v_m} - \tilde{s})$，而后在 $\tilde{P}$ 中用顶点序列 $\tilde{s}, \tilde{v_C}, \tilde{s'}$(或者 $\tilde{s'}, \tilde{v_C}, \tilde{s}$，取决于是否遵循逆时针顺序) 替换 $\tilde{v_m}$。如果这样的矩形是不可切的，那么就尝试让 $\tilde{s}$ 成为另外一个顶点，或者尝试剩下的交点。当 C 固定之后，Cut 操作是可能会失败的，就如图 4中的一样，这时应该重新随机选择一个单元格 C。算法过程如 1所示。

Algorithm 1: Inflate-Cut 算法

Input: 顶点数 n，$n \geq 4$ 且为偶数

Output: 一个随机的 n-格点正交多边形 P

```
r := n/2 − 2;
P := (1, 1), (1, 2), (2, 2), (2, 1);
while r > 0 do
    repeat
        随机选择 P 中的一个单元格 C;
        用 C 对 P 执行 Inflate 过程得到 P̃;
        对 P̃ 执行 Cut 过程，随机切割一个矩形;
    until Cut 成功;
    P := P̃;
    r := r − 1;
```

在生成完正交多边形后，随机其上的一条边作为灯塔问题的目标 t。

在整个生成算法中，需要线性的空间记录 P 的顶点坐标，并且维护有序的垂直边和水平边列表，维护每一行的格点数，注意到水平边和垂直边数各为 $\frac{n}{2}$，单元格的行数和列数各为 $\frac{n}{2} - 1$，因而空间复杂度为 $O(n)$。考虑从 n-正交多边形生成 $n+2$-正交多边形的过程，首先需要随机一个格点，因此要遍历水平边列表和垂直边列表，消耗 $O(n)$ 时间；执行 Inflate 操作，消耗 $O(n)$ 的时间；计算 C 的中心出发的射线和 P 的交点，消耗 $O(n)$ 时间；执行 Cut 操作，消耗 $O(1)$ 的时间；维护每一行的格点数及其他信息消耗 $O(n)$ 的时间。注意到这里如果采用一些高效的数据结构，可以将求交和维护两个方向的有序边列表的时间复杂度降到 $O(logn)$，但是随机格点和

Inflate 操作仍然需要 $O(n)$ 时间，因此意义不大。由于总共需要执行 n 次迭代过程，故整体需要 $O(n^2)$ 的时间生成 n-正交多边形。

2.2 直方图划分

考虑到目标边 t 同时也是求解结果中编号为 0 的灯塔（以下我们默认灯塔编号用非负整数表示），因此可以先对正交多边形进行一次直方图 (Histogram) 划分。直方图指的是以正交多边形 P 的内的一条线段为基 (base)，其余点坐标在 x 或 y 方向上单调连续的图形，类似于统计学中的直方图。直方图 h 的基边用 b_h 表示。将问题描述中的正交多边形 P 划分为多个直方图，如图 5所示。其中终点线段 t 就是直方图 A 的基边。

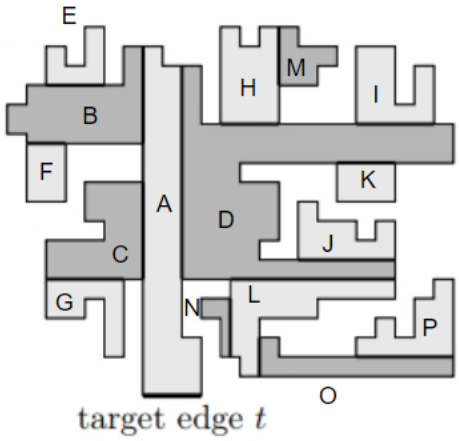

图 5: 直方图划分

从底边 t 开始的正交多边形可以用树状结构表示。将每个直方图对应于一个树节点，一个直方图 A 的基边若是另一个直方图 B 的某条边的一部分，那么该直方图 A 对应的节点是 B 的子节点。我们用一棵树来表示多边形的直方图划分，树的根即是以目标边 (target edge t) 为基边的直方图。将图 5中对应的直方图划分表示为划分树 T，如图 6所示。后续的算法也都以该划分树为基础进行。

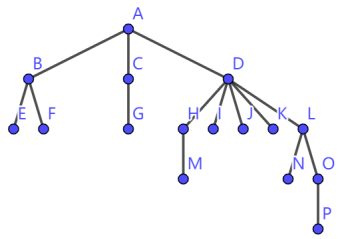

图 6: 直方图划分树

直方图划分的大致算法在 [2] 和 [4] 中都有大致的描述。以一条给定的边为基，对一个多边形进行直方图划分，主体分为两个步骤：首先，求出这条给定的基边向多边形内投影图形，该投影图形就是多边形内以该边为底边的直方图；其次，把这个直方图从多边形中截掉，然后对剩下的多边形进行递归式地求解，直到不可再分为止。

投影问题本身是一个相对复杂的问题，但在正交多边形的前提下，该问题可以通这对整个多边形扫一边，在扫描的过程中，用一个栈来维护直方图顶部的线段来进行求解。同样地，求解出直方图后，截去直方图的操作也可以通过扫描一边多边形，过和中按照各点与该直方图边界的关系，处理和记录剩余的多边形，一边扫描同样可以完成求解。两步的复杂度都是 O(n) 的，和多边形的边数呈正比例相关。而一条边最多同时属于两个直方图，因此最多被计算两次。因此，考虑上递归部分的复杂度，整个直方图划分的时间复杂度依然是 O(n) 的。

2.3 线段集合的生成与解的构造

我们通过一种基边上的线段结构来定义问题求解的搜索空间。对于任意直方图 h，定义位于基边上的线段对。下文图示中，我们用蓝色表示线段对中的第一个线段，红色表示线段对中的第二个线段，分别用 B_h 和 R_h 表示，因此线段对记为 (B_h, R_h)。线段 I_h 可以表示线段对中的任意一个线段。

由于我们已经生成了树形的直方图划分，我们定义线段在直方图树上的传递如下：考虑两个直方图 h 和 h'，其中 h 是 h' 在直方图树 T 中的孩子节点。由第 2.2节的定义，h 的基边一定在 h' 的某条边上。对于 h 上的某条线段 I_h，选择距离 $b_{h'}$ 更近的端点沿着与 $b_{h'}$ 平行的方向投影到 h' 的另一侧的边上，得到一个新的端点。将以上两个端点沿着与 $b_{h'}$ 垂直的方

向投影到 $b_{h'}$ 上。得到的新线段 $I_{h'}$ 即为线段 I_h 从 h 到 h' 的传递，记为 $I_{h'} = \mathrm{pr}(I_h)$。以上过程如图 7所示。

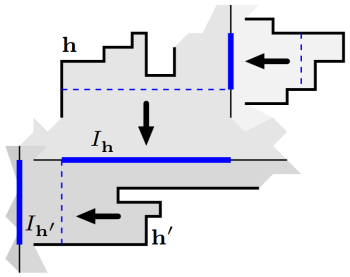

图 7: 基边上线段的传递

另外，我们对直方图上的两个特殊点进行定义。如图 8所示，对于某个直方图 h，基边 b_h 最右边的点定义为 q_h。从 q_h 开始，沿着 h 的边逆时针遍历，遍历过程中 x 和 y 方向上的坐标单调变化，达到极值时的点即为 v_h。在形状上看，q_h 和 v_h 就像组成了一个向上的阶梯。将 q_h 垂直投影到 b_h 上，得到的点记为 p_h。

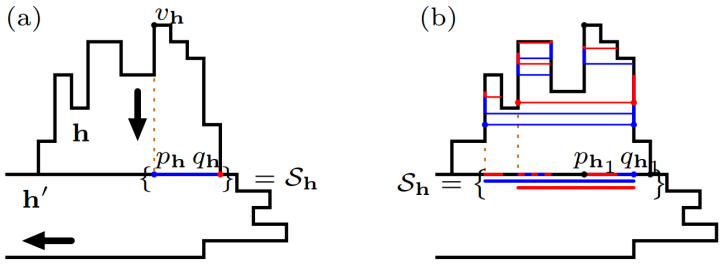

图 8: 直方图上的线段定义

对于每个直方图 h，定义线段对集合 S_h 如下：
对于 T 上的叶节点，$S_h = P_h$。其中 $P_h = \{[p_h, q_h], [q_h, q_h]\}$。
对于 T 上的非叶节点，有

$$S_h = \{\mathrm{pr}(B_{\overline{h}}), (\mathrm{pr}(R_{\overline{h}}))|(B_{\overline{h}}, R_{\overline{h}}) \in S_{\overline{h}}, \forall \overline{h} \in T_h\} \cup P_h.$$

其中 T_h 为 T 中直方图 h 的孩子节点集合。如果对于 h 的某个孩子 $\overline{h}$，存在 $(B_{\overline{h}}, R_{\overline{h}}) \in S_{\overline{h}}$，且 $\mathrm{pr}(R_{\overline{h}}) \cap [p_h, q_h] \neq \emptyset$，那么 $P_h = \emptyset$，否则 $P_h = \{[p_h, q_h], [q_h, q_h]\}$。我们给线段对中的每个线段赋一个 0 或 1 的值，所有其

他线段标为 1 值，如果存在线段 $[q_h, q_h]$，则标为 0 值。每个直方图的 S_h 可以递归求解，得到 h 的孩子节点线段对集合 $S_{\overline{h}}$ 后，S_h 可以在线性时间内求解，只需要遍历直方图边界即可。

求出 T 中每个 h 的 S_h 之后，我们就可以通过这些线段集合来构造 SLP 问题的解。我们定义每个直方图的线段对集合 S_h 的一个实现 (realization)，用 Γ_h 表示。Γ_h 是一个线段的集合，它包含 S_h 中每一个线段对中的至多一个线段。实际上，如果 Γ_h 中的每一个标记值为 1 的值都被一座灯塔穿过，那么相应的灯塔集合即为一个可行解（但不一定是最优解）。显然，构造 Γ_h 的可能性是指数级别的，我们需要成本更低的搜索和选择算法。

我们发现，不同的线段可能只需要构造一座灯塔，我们可能可以以此缩小灯塔解的数目。因此，得出解构造方法前，还需要给出直方图上线段的另外两个定义，如图 9所示。令 $I_{\overline{h}}$ 和 $I_{\hat{h}}$ 表示 h 的子直方图 $\overline{h}$ 和 $\hat{h}$ 上的线段，$I_h = \mathrm{pr}(I_{\overline{h}})$，$I'_h = \mathrm{pr}(I_{\hat{h}})$。如 (a) 图所示，选择 $I_{\hat{h}}$ 上距离 b_h 更近的点，和该点沿着 b_h 方向投影到直方图另一侧的边得到的点，切割出不包括基边的部分。如果得到的子直方图包含线段 $I_{\overline{h}}$，那么认为线段 $I_{\overline{h}}$ 支配 (dominates) 线段 $I_{\hat{h}}$。如果某个线段不被任何其他线段支配，那么将该线段称为的一个主线段 (master)。如 (b) 图所示，如果 $I_{\hat{h}}$ 上存在某个点，其沿着 h 基边的方向投影到直方图另一侧得到的点在 $I_{\overline{h}}$，那么我们认为 $I_{\hat{h}}$ 和 $I_{\overline{h}}$ 匹配 (match)。

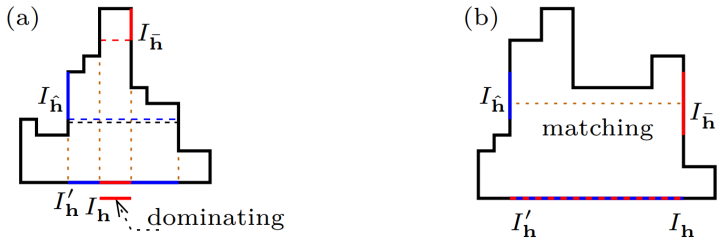

图 9: 直方图上的线段支配和匹配

至此，我们可以递归得出对应于某个直方图 h(相应的实现 Γ_h) 的 SLP 问题解的灯塔数，其中 V_{Γ_h} 是 h 中主线段的个数，$M\Gamma_h$ 是 h 中的独立匹

配线段集合的大小，这些线段最多和线段集合里的一条其他线段匹配：

$$s(\Gamma_h) = \begin{cases} 0 & \text{, if } h \text{ is a leaf and } \Gamma_h = \{[q_h, q_h]\} \\ 1 & \text{, if } h \text{ is a leaf and } \Gamma_h = \{[p_h, q_h]\} \\ V_{\Gamma_h} - M_{\Gamma_h} + \sum_{\overline{h} \in T_h} s(\Gamma_{\overline{h}}) & \text{, otherwise} \end{cases} . \tag{1}$$

文献 [3] 的后面部分简单描述了如何以灯塔数为基准，递归推导出 SLP 问题解的结果，这里不再赘述。然而，公式 (1) 存在正确性问题，实际上该文作者已经给出了反例证明，该灯塔数目结果并非最优解。

3 算法设计和优化

根据对已有工作算法描述的总结和思考，我们发现以下几个问题：

- 没有给出具体的灯塔生成的高效算法。
- 定理描述过于复杂，缺乏简洁而有效的证明，问题抽象形式较为晦涩，缺乏具象化描述，没有与具体问题的场景相联系。
- 部分特殊算例还存在正确性问题。

因此，我们对该算法进行了细致的研究和探索，整理并思考了算法思路，设计了一种更简单的算法，并更形象地将算法描述与实际场景相结合。同时，我们考虑了一些可能造成非全局最优解的情况。

首先，需要对目标正交多边形进行直方图分解。这里算法与上一章 2.2节的相同，得到直方图分解树 T。

要得到 SLP 问题的解，只需要按照某种规则与直方图的基线垂直建立灯塔。定义 h 的灯塔某个解集合为 l_h，l_h 应该保证为 h 所有的子节点 $\overline{h} \in T_h$ 提供导航。这种导航包括，与 $\overline{h}$ 的某个灯塔相交，或者覆盖灯塔集 $l_{\overline{h}}$ 未覆盖的 $\overline{h}$ 区域。在此，定义在 h 基线上的线段集合 s_h，该集合的含义为，对 T 中每个直方图 h，分别建立垂直于 b_h 的灯塔集合穿过 s_h 中的所有线段，即可得到相应的 SLP 问题的解。我们提出一种自底向上的递归算法来开展求解。

描述算法前，首先对传递进行重新定义。与上一章 2.3节中的传递不同，我们定义远端传递和近端传递。如图 10所示，直方图 h 的某个子直方图 $\overline{h}$ 上存在某个线段 $I_{\overline{h}}$(图中就是基边 $b_{\overline{h}}$)。远端传递为距离 b_h 较远的点，与其

沿着 b_h 方向在 h 另一侧上的水平投影点，在 b_h 上的垂直投影，即图中的线段 BC，记为 $\mathrm{rpr}(I_{\overline{h}})$；近端传递定义类似，区别是选择距离 b_h 更近的点，得到图中的线段 AC，记为 $\mathrm{npr}(I_{\overline{h}})$。我们认为，穿过远端传递的灯塔对 $I_{\overline{h}}$ 上的所有点可见，而穿过近端传递的灯塔对 $I_{\overline{h}}$ 上的一部分可见。

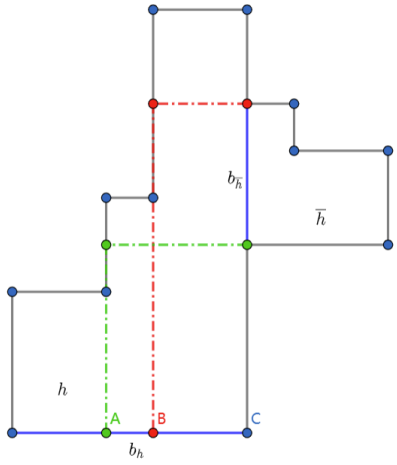

图 10: 远端传递和近端传递

定义直方图上的线段超集合 s_h^*。当 h 为叶子节点时，$s_h^* = \emptyset$，表示无需在该直方图中建立灯塔，因为我们认为其父节点的灯塔可以完成 h 区域的覆盖。

如果 h 为非叶子节点，且 $\overline{h} \in T_h$ 为叶子节点。建立所有 $\overline{h}$ 的基边 $b_{\overline{h}}$ 在基边 b_h 上的远端传递 $\mathrm{rpr}(b_{\overline{h}})$。对任何一个穿过该传递的灯塔，$\overline{h}$ 的内部区域对其可见，其中的点可以先到达 l_h 中的灯塔，再到达 b_h。将该传递加入线段超集合 s_h^*。

如果 h 为非叶子节点，且 $\overline{h} \in T_h$ 为非叶子节点，先寻找 $s_{\overline{h}}$ 中的最小匹配集。匹配的定义与上一章 2.3节的相同。对于匹配集中的任意一条线段 I，h 的子直方图 h 的所有线段集合 $s_{\overline{h}}$ 中最多有一条线段 $I_{\overline{h}}$ 与之匹配，最小匹配集是满足要求的集合中元素最少的。如图 11(a) 所示，直方图 h 有五个非叶子节点的子直方图，绿色线段代表子直方图线段集合 $s_{\overline{h}}$，我们对这个直方图生成最大匹配集。注意到，由于直方图的性质，子直方图 $\overline{h}$ 的区域要么位于 $b_{\overline{h}}$ 的左边，要么位于 $b_{\overline{h}}$ 的右边，下称左子直方图和右子直方图。左子直方图上的线段一定只能和右子直方图的线段匹配，反之亦然。因此，把 $s_{\overline{h}}$ 上的每个线段看作无向图中的一个顶点，最小匹配集的求解就可

以抽象为图匹配问题。左右子直方图的节点间一定不会连通，可以进一步简化为二分图匹配问题，如图 11(b) 所示。二分图匹配可以用著名的匈牙利算法 (见 [1]) 开展求解。匹配线段的识别和最小匹配集的生成可以用堆栈的方式，在 $O(n)$ 时间内完成。求解出最大匹配集后，我们用该集合对 $s_{\overline{h}}$ 上的线段进行截取，保证任意最大匹配集中的线段都是对彼此完全可见的。完全可见是指，线段沿着 b_h 方向在直方图另一侧的投影与其匹配线段重合，如图 11(c) 红色线段所示。

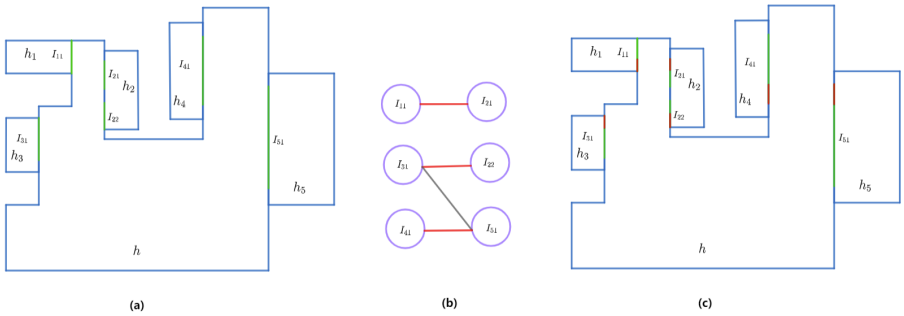

图 11: 最大匹配集的生成

匹配完成后，建立近端传递 $\mathrm{npr}(I_{\overline{h}})$。线段集合 $s_{\overline{h}}$ 是 $s_{\overline{h}}^*$ 的一个子集。那么，对任何一个穿过该传递的灯塔 l_1，存在一条与之相交且穿过 $I_{\overline{h}}$ 的灯塔 l_2，因此，所有可以到达灯塔 l_2 的点一定可以前往 l_1，然后到达 b_h。将该传递加入线段超集合 s_h^*。

s_h^* 集合中的主线段集合即为 s_h。主线段的定义与上一章 2.3节的相同，这里我们给出了一个更简单的判定方法。如果对于 s_h^* 中的某条线段 I，$\forall I' \in s_h^* - I$，要么 $I \cup I' \neq I'$，要么 $I = I'$ 且 I 对应的子直方图编号小于 I' 对应的子直方图编号 (不失一般性，我们可以对子直方图进行任意顺序的编号)，那么 I 为 s_h^* 的一个主线段。

我们可以由此需要找到最小数目的灯塔集合 l_h，使得 s_h 中的任何一个线段都与 l_h 中的至少一个灯塔相交。生成灯塔时，采用自上而下递归生成的方式。对于 T 中的某个直方图 h，对任意 $I_h \in s_h$，在 I_h 中的某个位置生成一座灯塔穿过 I_h，并与 h 的某条边 (非 b_h) 相交于 y 点。将 y 点沿着与 b_h 平行的方向投影在两条方向与 b_h 垂直的边上，如果投影点与 h 某个子直方图的 $s_{\overline{h}}$ 中的某线段 $I_{\overline{h}}$ 相交，那么截取交点与 $I_{\overline{h}}$ 与 b_h 距离更近的点，得到一条新线段，替换原 $s_{\overline{h}}$ 中的 $I_{\overline{h}}$ 线段，如图 12所示。由于每条线

段最多与直方图边界有两个投影交点，扫描和投影操作可以在 $O(n)$ 时间内完成。完成 l_h，递归生成 $l_{\overline{h}}$，最终所有直方图的灯塔集合的并集即为 SLP 问题的解。

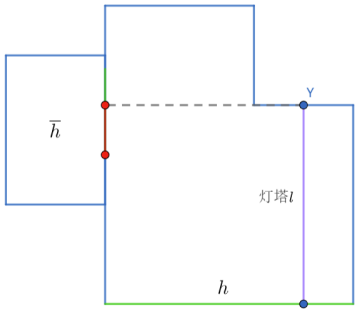

图 12: 灯塔生成与子直方图线段截取

当 $\overline{h}$ 为非叶子节点时，我们生成 s_h 会用到 $I_{\overline{h}}$ 的近端传递，这意味着穿过该传递的灯塔 l 不一定会对 $I_{\overline{h}}$ 对应的区域全部可见，除非穿过 $I_{\overline{h}}$ 的灯塔与 l 相交。因此，穿过 $I_{\overline{h}}$ 的灯塔位置要求会随着父节点灯塔 l 的位置变化。这种变化关系是需要通过自上而下的递归维护实现的。相比于 [3]，我们给出了一种更详细且可行的灯塔生成方法描述。

由于时间关系，我们实现了以上算法的一个简化版本。对于某个直方图 h 的子节点 $\overline{h}$，直接采用其基边 $b_{\overline{h}}$ 代替 $s_{\overline{h}}$ 进行传递。匹配的线段并没有进行合并和截取，这可能造成匹配线段间灯塔的部分冗余。我们认为，我们算法实现可以高效地在 $O(n^2)$ 时间内给出一个 2-近似的可行解。

4 数据结构与模块划分

4.1 数据结构

在测例生成的过程中，多边形用一个双向链表表示，它逆时针记录所有顶点的坐标，水平、垂直有序边列表用 vector 表示，每一行的单元格数目也用 vector 表示。

在直方图分解的过程中，对于多边形和直方图，我们在实现时使用同一种数据结构来表示。一个整型 n 表示顶点数，一个 vector 以逆时针的顺序存储多边形中所有顶点的坐标，需要说明的是，为了方便过程中的处理，实际存储了 n+2 个点的信息，第 1 和第 n 个点的信息分别被在第 n+1 和第

0 个位置多存储了一遍。对于直方图，额外记录了该直方图在直方图分解树中的相关信息，其 id，父结点以及子结点的 id。

灯塔则是用一个相对简单的数据结构表示，仅记录该灯塔两个端点的坐标。

4.2 模块划分

模块整体上可以划分为前端和后端。前端主要使用 d3.js 进行可视化，后端算法部分使用 C++ 实现。后端算法上又可以分为三个模块，分别实现了随机正交多边形的生成、正交多边形的直方图分解，以及最终的灯塔问题求解。

5 吞吐能力与性能评估

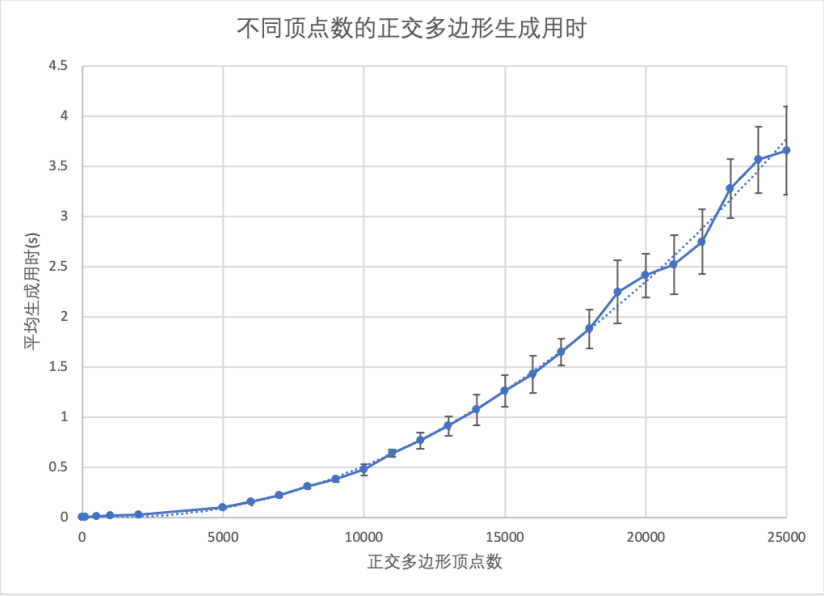

图 13: 测试用例生成用时

在这一部分我们着重对测试用例的生成、直方图的划分以及整个灯塔问题的求解 (包括直方图划分) 分别做了性能测试。对于测试用例的生成，测试给定不同大小的 n 时所花时间；对于后两者，将测试用例作为输入，测试

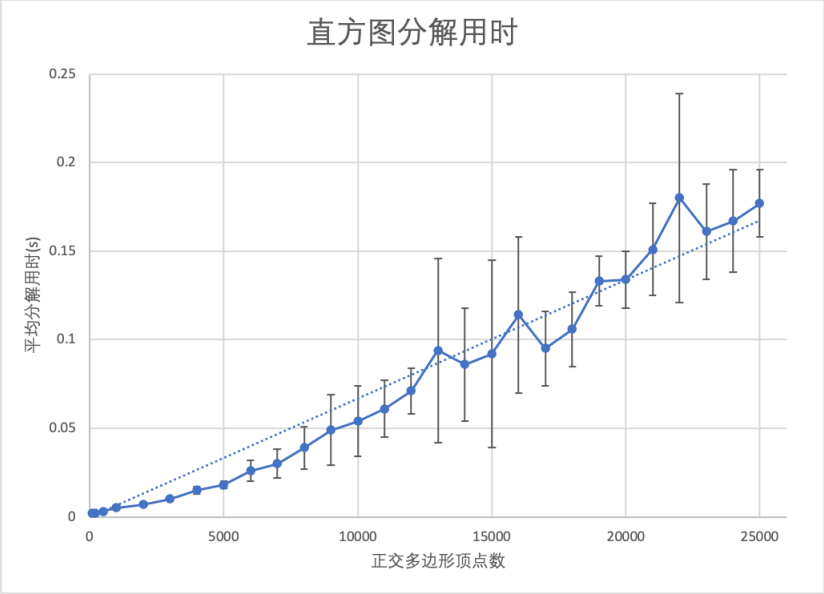

图 14: 直方图划分用时

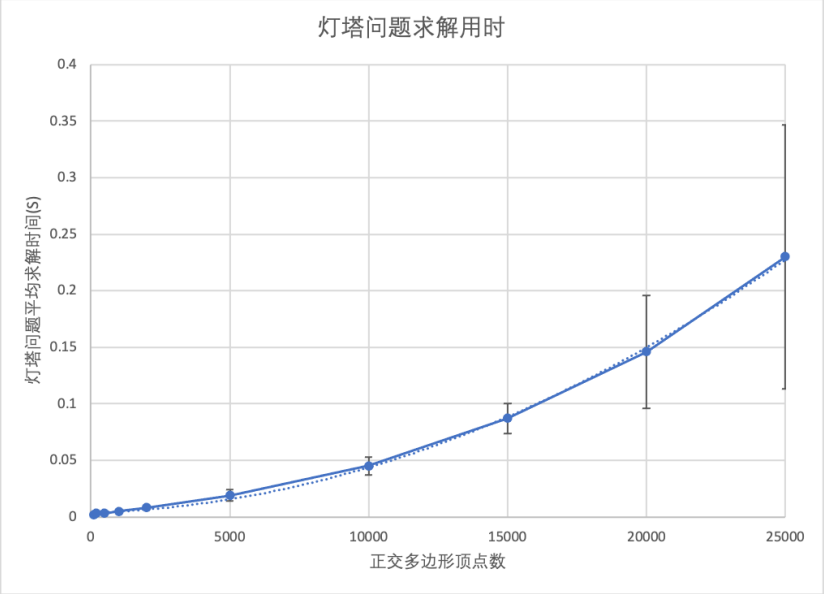

图 15: 灯塔问题求解用时

不同输入规模下花费的时间，对于同样规模的数据，每次测试均随机 10 次计算平均值和标准差。三者的性能测试结果如图 13,14, 15所示，标准差用误差线表示。可以看到随着数据规模的增加，各个模块用时的上升趋势符合我们对于其复杂度的估计，即测试用例生成 $O(n^2)$，直方图划分 $O(n)$，整个求解过程 $O(n^2)$。

在前端进行展示时，为了使得画面的可视化效果较好，限制了 $n \leq 1000$。为了方便性能的测试，也打包了单独用于测试大规模数据的二进制文件，具体参照用户手册。

6 小组分工

琚锡廷: 直方图分解，算法调研，简化的灯塔问题求解算法实现，报告撰写。

刘应天: 前端绘制，算法调研，测例生成，性能测试，报告撰写。

胡正丁: 前端绘制，算法调研，算法理论设计，报告撰写。

7 实验总结

本次实验中，我们选择了带导航的灯塔问题的最新工作成果作为研究对象。该问题的场景描述并不复杂，其求解却存在巨大挑战。首先，该问题需要对给定正交多边形进行直方图分解，得到直方图分解树。[3] 一文中的作者定义了一种新颖的线段在直方图树上的投影传递方法，并以此为基础给出了 $O(n^2)$ 时间的 SLP 问题的递归解法。然而，该文的部分定理已经被发现存在错误，因此该解法无法适用于所有情况。我们在调研相关论文的过程中受到启发，对线段投影，传递和合并方法进行了重新描述，并给出更详细的灯塔生成算法。我们虽然未严格证明新提出算法的正确性，但我们认为，相比于原文作者，我们的工作在完整性和描述形式上有了一定的改进。在实际程序中，我们实现了一个相对简化的版本，并给予充分的可视化支持。我们的程序具有良好的运行效果，可以在短时间内给出可行解，前端程序会详细展示从直方图生成到灯塔建立，再到给定点的导航，充分体现算法的推进流程。此外，我们还调研了正交多边形生成的相关工作并完成了实现，这保证了我们实验算例的普遍性和科学性。

在算法调研、设计和实现的过程中，我们充分借鉴了已有工作，较深入

地探讨和总结了该问题的思路，对一系列相关问题有了全新的认识，收获很大。在写程序的过程中，我们也充分学习并掌握了算法具体实现，前后端交互和前端数据可视化的部分知识。在整个实验中，我们也体会到沟通与交流能力的重要性，学习到多人合作学习和代码开发的技巧，最终完成了一份我们相对满意的工作。

参考文献

[1] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics*, 52(1-2):7–21, 2010.

[2] C. Levcopoulos. Heuristics for minimum decompositions of polygons. *Linkoping University*, 1987.

[3] Bengt J Nilsson and Paweł Żyliński. The lighthouse problem.

[4] Jörg-Rüdiger Sack and Jorge Urrutia. *Handbook of computational geometry*. Elsevier, 1999.

[5] Ana Paula Tomás and António Leslie Bajuelos. Generating random orthogonal polygons. In Ricardo Conejo, Maite Urretavizcaya, and José-Luis Pérez-de-la-Cruz, editors, *Current Topics in Artificial Intelligence, 10th Conference of the Spanish Association for Artificial Intelligence, CAEPIA 2003, and 5th Conference on Technology Transfer, TTIA 2003, San Sebastian, Spain, November 12-14, 2003. Revised Selected Papers*, volume 3040 of *Lecture Notes in Computer Science*, pages 364–373. Springer, 2003.