arXiv:2303.07858v1 [cs.CG] 14 Mar 2023

高效 Yao 图构造

DanielFunke ！ 卡尔斯鲁厄

理工学院**Peter Sanders** ✉

卡尔斯鲁厄理工学院

摘要

Yao 图是一种几何伸展图，它将给定点集中的每个点连接到以该点为中心绘制的每个 k 锥中的最近邻。Yao 图最初用于在 d 维空间中构造最小生成树。此外，它们还用于无线网络中的拓扑控制等场景。文献中已提出一种最优的 $\mathcal{O}(n\log n)$ 时间算法，可为给定点集构造 Yao 图，但据我们所知，该算法从未被实现。相反，常用软件包使用复杂度为二次的算法来构造这些图。本文介绍了首个最优 Yao 图算法的实现。我们开发并调优了实现 $\mathcal{O}(n\log n)$ 界所需的数据结构，并详细说明了将原始算法从理论应用于实践所需的算法调整。我们提出了一种优先队列数据结构，将静态事件与动态事件分离，这一结构也可能对其他扫描线算法具有独立价值。此外，我们还提出了一种基于均匀网格数据结构的新型 Yao 图算法，在中等规模的输入上表现良好。我们在广泛的合成数据集和真实世界数据集上评估了这些实现，结果表明，我们的实现比目前公开可用的实现至少快一个数量级。

2012年 ACM 学科分类计算理论 → 稀疏化与伸展图

关键词与短语计算几何，几何伸展图，Yao 图，扫描线算法，最优算法

补充材料源代码可在以下位置获取 `https://github.com/dfunke/YaoGraph`

1 引言

Yao 图是一种几何伸展图，它将给定点集中的每个点与其在每个 k 个锥中的最近邻相连，示例见图 1。t-伸展图是一种加权图，对于任意一对顶点，它们之间都存在一条 t-路径，其权重至多为它们空间距离的 t 倍。参数 t 称为伸展图的伸展因子。对于 $k > 6$，Yao 图的伸展因子可界定为 $1-2\,{}^{1}\!/_{\text{正弦}\,\pi/k}$ [10]。Yao 提出这类图，用于在 d 维空间中构造最小生成树 [13]。此外，这类图还用于无线网络中的拓扑控制等场景 [11]。Chang 等人 [5]提出了一种最优算法，可在$\mathcal{O}(n\log n)$ 时间内构造这类图。由于该算法结构复杂，且依赖开销高昂的几何构造，据我们所知，目前尚无该算法的实现。相反，流行的 CGAL 库中的基于锥的伸展子图软件包采用了一种$\mathcal{O}\left(n^2\right)$ 界较差的算法 [12]。

贡献。 本文介绍了首个公开发布的 Chang 等人提出的 Yao 图构造最优算法实现。我们将该算法从理论付诸实践，开发了实现 $\mathcal{O}(n\log n)$ 界所需的数据结构，并详细描述了原论文中未提及的所有算法操作，例如输入点排序、复合边界处理以及包围区域搜索。在我们的事件队列中，我们将静态（输入点）事件与动态（交点）事件分开。这大幅提高了优先队列操作的效率，也可能成为其他扫描线算法的一项有用技术。

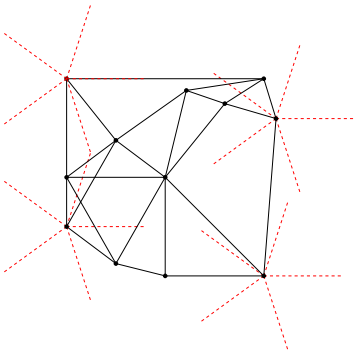


图 1 包含十个点和 $k = 5$ 个锥的 Yao 图。图中以红色虚线展示了围绕四个示例点的五个锥。

我们在各种合成数据集和真实世界数据集上测试了该算法。结果表明，尽管该算法结构复杂且使用了开销较大的几何构造，我们的实现仍比目前其他可用实现快一个数量级。此外，我们基于均匀网格数据结构开发了一种新的 Yao 图算法，该算法易于实现、便于并行化，并且在中等规模的输入数据上表现良好。

概要。 第 2 节回顾 Yao 图构造方面的相关工作。第 3 节介绍 Chang 等人的最优算法，以及实现该算法所需的调整。第 4 节将介绍更多实现细节，例如数据结构和几何运算。第 5 节评估我们的实现，并将其与其他方法进行比较。第 6 节总结本文，并展望未来工作。

定义。给定二维欧几里得空间中的点集 P 和整数参数 $k > 1$，Yao 图 $G_k = (\mathrm{P}, \mathrm{E})$ 是一个有向图，它将每个点$p \in \mathrm{P}$ 与每个 k 锥内的最近邻相连。每个锥 $\mathrm{C}_i = (\theta_L, \theta_R)$，$0 \le i < k$，由两条夹角分别为 $\theta_L = \frac{2i\pi}{k}$ 和 $\theta_R = \frac{2(i+1)}{k}\pi$ 的边界射线定义。我们将以点 p 为顶点的锥 C_i^p 记为 C_i。此外，我们规定，角度为 θ_L 的左侧（即逆时针方向）边界射线属于锥 C，而右侧边界射线不属于该锥。也就是说，对于给定点 $p \in \mathrm{P}$ 和锥 C_i^p，我们定义点集 $\mathrm{P} \cap \mathrm{C}_i^p := \{q \in \mathrm{P} : \sphericalangle p\,q \in [\theta_L, \theta_R)\}$。于是，Yao 图 $G_k = (\mathrm{P}, \mathrm{E})$ 的边集 E 可形式化定义为 $\mathrm{E} := \{\forall i \in [0, k)\ \forall p \in \mathrm{P} : (p, q) \text{ with } q = \arg\min^{v \in \mathrm{P} \cap \mathrm{C}_i^p}(d(p, v))\}$，其中 $d(\cdot, \cdot)$ 表示欧几里得距离函数。

2 相关工作

Yao 提出了一种$\mathcal{O}(n^{5/3} \log n)$ 时间算法，用于求解八邻居问题 ——即构造一个含有 $k = 8$ 的 Yao 图。该算法基于将欧几里得空间划分为单元的镶嵌。对于给定的点和锥，每个单元都会被判定是否可能包含最近邻候选点，从而减少必要的距离计算次数。Chang 等人给出了该问题的最优解，并提出了一种$\mathcal{O}(n \log n)$时间算法，用于构造给定点集及参数 k [5] 对应的 Yao 图。他们的算法遵循 Fortune 算法的相同结构，用于构造点集 [8], 的 Voronoi图

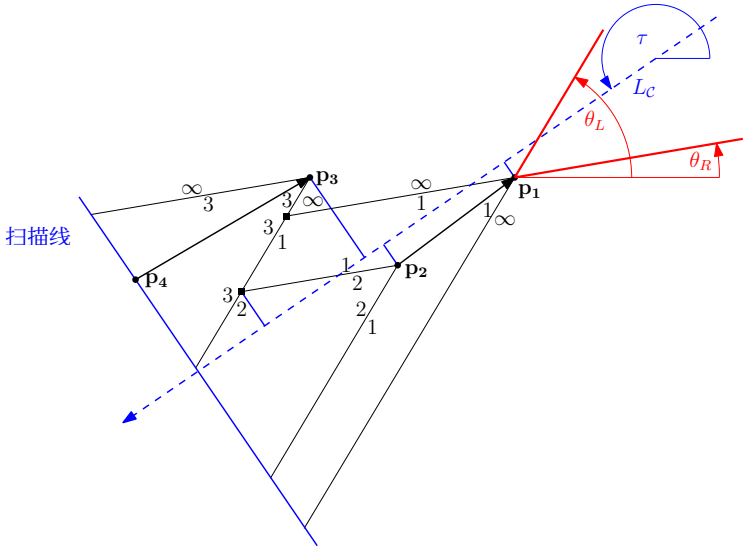


图 2 锥 $\mathcal{C} = (\theta_L, \theta_R)$（以红色标出）的扫描线算法示例状态。输入点（圆形、粗体标签）按扫描线扫过的顺序编号，其在锥内角平分线上的投影以蓝色显示。射线标注了它们所分隔的区域。相交事件以方形标出。已确定的 Yao 图边以箭头表示。

该算法采用 Bentley 和 Ottmann 最初提出的扫描线技术来计算线段交点 [3]。然而，尽管已有许多 Fortune 算法的实现，但据我们所知，Chang 等人的 Yao 图算法尚无实现。例如，CGAL 库的基于锥的伸展子图软件包实现了一种效率较低的$\mathcal{O}(n^2)$ 时间算法 [12]。他们的算法是对一种用于构造 Θ 图 [10] 的扫描线算法的改编。 Θ 图的定义与 Yao 图类似，但每个锥中的最近邻并非按欧几里得距离定义，而是按其在锥的内角平分线上的投影距离定义。这使得可以使用$\mathcal{O}(n\log\ n)$扫描线算法，并以平衡搜索树作为扫描线数据结构来回答一维范围查询 [10]。对于 Yao 图，这种降维并不可行，因此 CGAL 的算法会在线扫描线数据结构中进行线性搜索以查找最近邻，导致其复杂度为$\mathcal{O}(n^2)$ 。不过，CGAL 的算法比 [5] 提出的最优算法更易于实现，而且不需要几何构造，只需要谓词。第 4 节的表 1 概述了两种算法所需的几何运算。

Yao 图在无线网络中的应用需要采用不同的技术和算法，因为网络中的每个点（即节点）只能部分了解所有可用站点。因此，需要由节点利用其部分信息在本地计算这些网络的 Yao 图。Zhang 等人提出了一种适用于此情形的算法，该算法还考虑了无线电干扰造成的扰动 [14]。

3 算法

在 1990 年的论文中，Chang 等人提出了一种 O($n\log\ n$) 时间的扫描线算法，用于计算点集的有向 *Voronoi* 图 。通过稍作修改，他们的算法可以在以下范围内计算点集的地理邻域图 （或 Yao 图）

■ 算法 1 锥（θ_L, θ_R）的扫描线算法。 $L(p, \theta, R_a, R_b)$ 和 $L(\overleftrightarrow{pv}, R_a, R_b)$ 分别表示起点为 p、角度为 θ 的射线，以及从 p 到 v 的线段，二者均分隔区域 R_a 和 R_b。输入：点集 P $= \{p_1, \ldots, p_n\}$，其中 $p_i \in \mathbb{R}^2$，锥 (θ_L, θ_R) 输出：GNG $G = (V, E)$ 1：锥的内角平分线的反向 2： $Q \leftarrow \{(\rho_\tau(p) \quad p, I) : p \in$ P$\}$. 用输入点初始化优先队列 3： $SL \leftarrow \emptyset$ 4： $G = (V, E)$ $\leftarrow$ (P $\emptyset$) 5：当 $p \leftarrow$ 弹出操作 popMin(Q) 时执行 6：如果 p 是输入点，则 7：B_L, $B_R \leftarrow$ 查找区域findRegion(p, SL) $\triangleright$ B_L，并且 B_R 包围 p 8：$E \cup= (p, B_L^r)$. 断言 $\left(B_L^r == B_R^l\right)$ 9：如果 $B_L \cap B_R = v$，则从 Q 中删除 v 10：$B_L^* \leftarrow L(p, \theta_L + \pi, B_L^r, R_p)$ 11：$B_R^* \leftarrow L\left(p, \theta_R + \pi, R_p, B_R^l\right)$ 12：将 $[B_L^*, B_R^*]$ 插入操作到 SL 中，位于 B_L 和 B_R 之间 13：如果 $B_L \cap B_L^* = v$，则 $Q \cup= (\rho_\tau(v) \quad v)$ 14：如果 $B_R \cap B_R^* = v$，则 $Q \cup= (\rho_\tau(v) \quad v)$ 15：如果 p 是交点，则 16：$B_L, B_R \leftarrow$ 在 p 处的相交射线 17：$a \leftarrow B_L^l \quad b \leftarrow B_R^r$ 18：如果 $B_L \cap$ prev(B_L) $= v$，则从 $Q \quad \triangleright$ 中删除 v，即 SL 上的左邻域边界 19：如果 $B_R \cap$ succ(B_R) $= v$，则从 $Q \quad \triangleright$ 中删除 v，即 SL 上的右邻域边界 20：$L_{BS} \leftarrow L(\frac{\vec{a}+\vec{b}}{2}, \sphericalangle ab + \frac{\pi}{2}, R_a, R_b)$. 线段 $\overleftrightarrow{ab}$ 的平分线 21：$L_L \leftarrow L(p, \theta_L + \pi, R_a, R_b)$ 22：$L_R \leftarrow L(p, \theta_R + \pi, R_a, R_b)$ 23：如果 $L_L \cap L_{BS} = \emptyset = L_{BS} \cap L_R$，则 . 平分线不与来自 p 的任何直线相交 24：$B^* = L\left(p, \pi + \begin{cases} \theta_L \\ \theta_R \end{cases}\right.$ 如果 $\rho_\tau(a) < \rho_\tau(b)$，否则 $\left. R_a, R_b \right)$

25：如果 $L_L \cap L_{BS} = p = L_{BS} \cap L_R$，则 . 平分线与 p 中的两条直线都相交 26：$B^* = L(p, \sphericalangle ab + \frac{\pi}{2}, R_a, R_b)$ 27：如果 $L_L \cap L_{BS} = v$ 或 $L_{BS} \cap L_R = v$，则 . 平分线与 v 中的一条直线相交 28：$B^* = L(\overleftrightarrow{pv}, R_a, R_b) + L(v, \sphericalangle ab + \frac{\pi}{2}, R_a, R_b)$ 29：$Q \cup= (\rho_\tau(v) \quad v)$. 删除事件 30：将 SL 中的 $[B_L, B_R]$ 替换为 B^* 31：如果 $B^* \cap$ prev(B^*) $= v$，则 $Q \cup= (\rho_\tau(v) \quad v)$ 32：如果 $B^* \cap$ succ(B^*) $= v$，则 $Q \cup= (\rho_\tau(v) \quad v)$ 33：如果 p 是删除点，则 34：属于 $p \quad \triangleright B = L(\overleftrightarrow{pv}, R_a, R_b)$ 的射线 $+ L(v, \sphericalangle ab + \frac{\pi}{2}, R_a, R_b)$ 35：将 SL 中的 B 替换为 $L(v, \underline{\sphericalangle a}b + \frac{\pi}{2}, R_a, R_b)$ 36：返回 G

相同的最优界 [5, 定理 3.2，定理 4.1].[1] 要构造点集 $\mathbf{P}$ 的 Yao 图$G_k = (\mathbf{P},\mathbf{E})$，需要进行 k 次扫描线遍历，每次考察一个特定的锥 $\mathcal{C} = (\theta_L, \theta_R)$。锥 $\mathcal{C}$ 的扫描线沿着与其内角平分线相反的方向推进，该平分线表示为一条经过原点、角度为 $\tau = \frac{\theta_L+\theta_R}{2} + \pi$ 的直线 $L_{\mathcal{C}}$。参见图 2 中的蓝色虚线。按照输入点在 $L_{\mathcal{C}}$上的投影排序，依此顺序扫描输入点

$$\rho_\tau(p) := \frac{\vec{p} \cdot \vec{SL}}{\vec{SL} \cdot \vec{SL}} \quad \forall p \in \mathbf{P} \text{ with } \vec{p} = (x, y) \text{ and } \vec{SL} = (\cos\tau, \sin\tau). \tag{1}$$

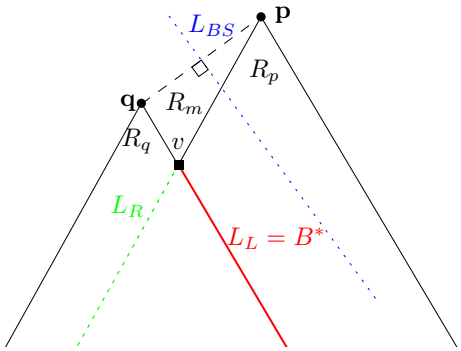
所有输入点都被插入事件优先队列 Q，其优先级为 $\rho_\tau(p)$，事件类型为 输入点。每个输入点 p 都是一个锥 $\mathcal{C}^p$ 的原点，该锥的边界射线为 B_L 和 B_R。锥 $\mathcal{C}^p$ 定义了平面上的区域 R_p；对于在 p 之后被扫描的任意点，点 p 都是相对于锥 $\mathcal{C}^p$ 的最近邻。算法的不变式是：一个点一旦被扫描，其在锥 $\mathcal{C}$ 中的最近邻就已确定。例如，在图 2 p_2 中，p_1 位于 p_1 的区域内，因此相对于锥 $\mathcal{C}$，p_2 的最近邻是 p_1。边界射线 $B_\square$ 总是将两个输入点的区域分隔开，因此由其原点 $B^p_\square$、角度 $B^\theta_\square$ 以及左右区域 $B^l_\square$ 和 $B^r_\square$ 定义。任意点的锥之外的区域均标记为无穷大。算法维护一个有序数据结构 SL，用于存储当前与扫描线相交的射线。这些射线按从左到右的顺序排列，该数据结构需要支持插入、删除和查找操作，以及访问给定射线的左右邻居。数据结构的细节见第 4.2 节。附录中的图 11 展示了本算法的执行示例。

3.1 事件类型

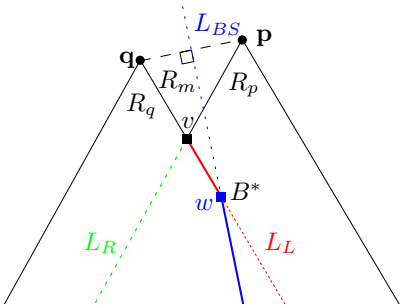
共有三种事件类型：1）输入点，2）交点，3）删除点[2]。下文将概述算法如何处理每种事件。详情见算法 1。如果多个事件重合，其处理顺序可以任意安排。

1) 输入点。给定集合 P 中的所有点都会在算法开始时插入事件优先队列。对于关联点为 p 的输入点事件，会在扫描线数据结构 SL 中查找包含 p 的区域 R_q。该区域由两条边界射线 B_L 和 B_R 及其关联区域 $B^r_L = B^l_R = R_q$ 定义。[3] 随后，我们可以将边 (p, q) 添加到 G_k 的边集 E 中，正如引理 3.1] 中所证明的那样（见 [5, ）。点 p 是区域 R_p 的顶点；该区域左侧以 $B^*_L = (p, \theta_L + \pi)$ 为界，将区域 R_q 和 R_p 分隔开来，右侧以 $B^*_R = (p, \theta_R + \pi)$ 为界，将区域 R_p 和 R_q 分隔开来。这些新射线会插入扫描线数据结构 SL 中，位于 B_L 和 B_R 之间，从而形成序列 $[B_L, B^*_L, B^*_R, B_R]$。最后，需要处理所考虑射线的交点。如果 B_L 和 B_R 在点 v 相交，则需要从优先队列 Q 中删除其对应的交点事件，因为 B_L 和 B_R 不再是相邻射线。取而代之的是，将 B_L 与 B^*_L 之间以及 B^*_R 与 B_R 之间可能存在的交点添加到 Q，以供后续处理。

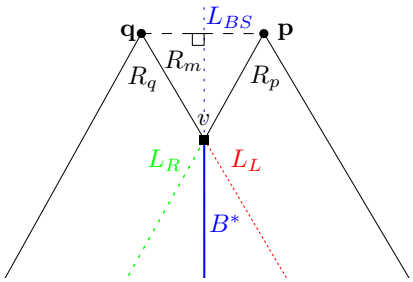
[1] 我们的实现计算 Yao 图，但也可以轻松修改为计算有向 Voronoi 图。[2] Chang 等人的原始算法描述中没有删除点 [5]。[3] 请注意，Chang 等人 [5] 会在扫描线数据结构中显式存储射线和区域。不过，由于一个区域可以通过其两条边界射线明确识别，我们选择了这种更简单的扫描线状态表示方式。



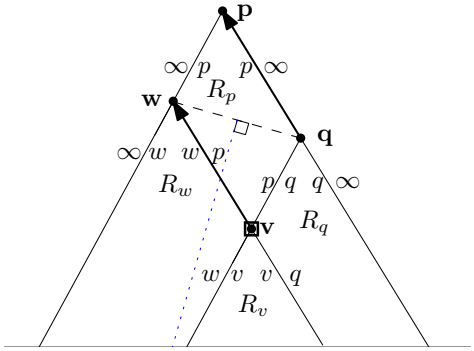
(a) L_{BS} 既不与 L_L 相交，也不与 L_R相交。



(b) L_{BS} 在 w中与 L_R 相交。



(c) L_{BS} 在 v中同时与 L_L 和 L_R 相交。



(d) 位于锥边界上的输入点。

图 3 交点事件发生后，边界 B^* 可能出现的三种配置示意图。在所有示例中，输入点 p 均先于 q被扫描。直线 L_L、L_R 和 L_{BS} 以虚线表示，生成的边界 B^* 以粗体表示。在图 d）中，Yao 图的边以粗箭头表示。

2) 交点。交点 v 与两条相交射线 B_L 和 B_R 相关联。它们分隔区域 $R_p := B_L^l$，$B_L^r = R_m = B_R^l$ 和 $B_R^r =: R_q$，参见图 3。区域 R_m 在交点 v 处终止，而从 v 开始出现一条位于 R_p 和 R_q 之间的新边界 B^*。B^* 的形状取决于点 p 和 q 的配置，可以是简单射线，也可以是线段与射线的并集。第 3.2 节将详细说明如何确定 B^*。随后，B^* 会在扫描线数据结构中取代序列 $[B_L, B_R]$。同样，需要处理所考虑射线的交点。如果 B_L 与其左邻射线有交点，或者 B_R 与其右邻射线有交点，则需要从事件队列 Q 中删除对应的交点事件。相应地，如果 B^* 与其左邻射线或右邻射线相交，则将相应的交点事件添加到 Q。

3) 删除事件。Chang 等人在论文中描述的原始算法并未包含删除事件，因为作者没有说明如何处理复合边界。我们使用删除事件来实现由一条线段和一条射线构成的边界。删除事件标记线段的结束和射线的开始。它不会改变扫描线数据结构的实际状态。

3.2 边界判定

如上一节所述，在交点事件 v处，两条相交射线 B_L 和 B_R 合并为一条新的边界 B^*，将区域 $R_p := B_L^r$ 和$R_q := B_R^l$分隔开。边界的形状由点 p 和 q的配置决定。具体判断依据是直线之间交点的数量

- $L_L = (v,\ \theta_L + \pi)$（绿色），$L_R = (v,\ \theta_R + \pi)$（红色），以及直
- 线 $\vec{pq}$, $L_{BS} = (\frac{\vec{p}+\vec{q}}{2},\ \sphericalangle pq + \pi/2)$ 的平分线 L_{BS}（蓝色虚线）。

这些颜色对应图 3 中的直线，图 3 展示了不同情况。如果 L_{BS} 既不与 L_L 相交，也不与 L_R 相交，那么若 p 先于 q被扫描，则为 $B^* = L_L = (v,\ \pi + \theta_R)$；否则为 $B^* = L_R = (v,\ \pi + \theta_R)$。直观地说，较低点相对于扫描线方向的区域会继续延伸，而上方区域则在交点 v处终止。如果 L_{BS} 同时与直线 L_L 和 L_R相交，那么两个交点必定与 v重合。在这种情况下，$B^* = (p,\ \sphericalangle pq + \pi/2)$，也就是说，边界沿着从点 v出发的平分线方向继续延伸。否则，也就是说，若 L_{BS} 与 L_L 或 L_R 中的一条直线在点 w相交，则所得边界 B^* 将由线段 $v\vec{w}$ 和射线（$w,\ \sphericalangle pq + \pi/2$）组成。在这种情况下，会在点 w处向优先队列添加一个删除点事件。图 3 展示了这三种情况。

与锥边界共线的输入点。 Chang 等人假设，任意两个输入点之间的连线都不与角度为 θ_L 或 θ_R的方向一致。下文将取消这一要求。位于角度为 $\theta_\square$ 的同一条直线上的输入点彼此<加粗>可见 。这会影响经过的边界射线所分隔的区域。以下讨论的图示请参见图 3d。回顾一下，角度为 θ_L 的<加粗>左侧 ——即逆时针方向——边界射线属于锥 $\mathcal{C}$，而右侧 边界射线则不属于该锥。如果边界射线 $B_\square = (p, \theta_\square, B_\square^l, B_\square^r)$与输入点 q相交，那么 $B_\square$ 会在 q 处终止，并形成一条新的边界射线 $B'_\square$ 。如果 $B_\square$ 是左边界，即 $\theta_\square = \theta_L$，则将边 (p, q) 添加到 G_K ，并且 $B'_\square$ 分隔区域 $B_\square^l$ 和 R_q。如果它是右边界，则不向 G_k 添加边，并且 $B'_\square$ 分隔 R_q 和 $B_\square^r$。图 3d 中的示例点为 q 和 w 。

3.3 分析

扫描线算法处理的事件总数，是输入点事件 N_{input}、交点事件 N_{IE} 和删除事件 N_{DE}的总和。为了界定扫描线算法处理的事件数量，我们考虑算法执行期间扫描线数据结构中可能存在的射线数量。

每个输入点事件都会向扫描线数据结构中添加两条射线 SL，因此总共有 $2n$ 条射线。它可能会从事件队列中删除一个交点事件 Q ，并且最多可能添加两个此类新事件。每个交点事件都会从SL 中删除两条射线，并添加一条新射线，因此扫描线的大小减少一。因此，在所有射线都从扫描线中删除之前，最多可以处理 $2n$个交点事件。一个交点事件除了删除自身外，还可能从Q 中删除另外两个交点事件，并可能添加一个新的交点事件。[4] 此外，还可能向 Q添加一个删除事件。因此，最多可以处理 $2n$ 个删除事件，每个事件都不会改变

[4] 严格来说， B^* 可以与它的两个邻居都相交，从而产生两个交点事件。然而，当第一个交点事件（按 $\rho_\tau(\cdot)$ 的定义）被处理时，它会从 Q中删除第二个事件，因为 B^* 会从 SL 中删除，而第一个事件会插入一条新的边界射线。

射线和交点的数量。总计，

$$\begin{aligned} N_{\text{events}} &\leq N_{\text{input}} + N_{\text{IE}} + N_{\text{DE}} \\ &\leq n + 2n + 2n = 5n \end{aligned} \tag{2}$$

借助合适的数据结构，可以在 $\mathcal{O}(\log n)$ 时间内处理每个事件，从而得到 $\mathcal{O}(n\log\ n)$ 的时间界。

4 实现细节

本节重点介绍我们实现 Chang 等人的 Yao 图算法时所做的一些设计决策。

4.1 几何内核

表 1 Yao 图构造算法中所用几何谓词的比较

	Chang 等人	CGAL	朴素方法	Grid
复杂度	$\mathcal{O}(n\log\ n)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
谓词				
欧几里得距离比较	X	X	X	X
到直线的距离比较	X	X		
直线的有向侧	X		(X)	(X)
构造				
锥边界	X	X	X	X
盒构造				X
直线投影	X			
射线相交	X			

Chang 等人的算法 [5] 需要许多不同的几何谓词和构造。我们以非精确 方式实现了所需谓词和构造的自定义版本。此外，用户也可以使用 CGAL 库提供的核。EPIC 核提供精确谓词和非精确构造，而 EPEC 核则具有精确谓词和精确构造 [4]。

表 1 列出了本文介绍的不同 Yao 图构造算法所使用的几何谓词。如果锥的边界射线是精确构造的，那么朴素方法和基于网格的 Yao 图算法都需要使用有向直线侧谓词，以确定点 q 相对于点 p位于锥C_I^p 的哪一侧。此外，网格算法可以通过精确计算来构造网格数据结构。然而，在我们的实现中，我们仅使用非精确计算将输入点放入网格单元，因为在所有实验中，我们都没有发现需要进行精确计算。请注意，判断一个网格单元是否可能包含比当前已找到的邻点更近的点，是通过（精确的）欧几里得距离比较谓词完成的。

为了减少代价高昂的射线—射线相交计算次数，我们将所有已发现的交点存储在哈希表中，并以两条相交射线作为键，参见例如算法 1::13。若要检查两条射线是否相交（例如算法 1::9），只需查询哈希表。

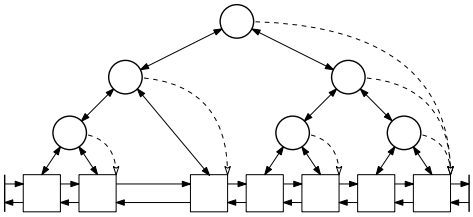


图 4 扫描线数据结构是一个由射线构成的双向链表，其上方配有一棵 AVL 搜索树。此外，每个节点都指向其子树中最右侧的射线（虚线）

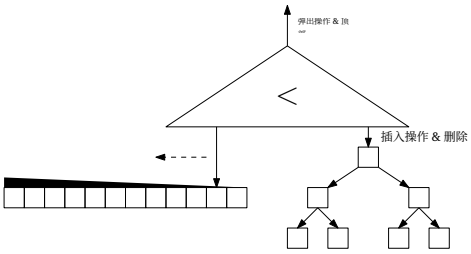


图 5 优先队列由一个静态的输入点有序数组，以及一个动态、可寻址的优先队列组成，用于处理交点和删除事件。

4.2 扫描线数据结构

Chang 等人证明了该算法的时间复杂度为$\mathcal{O}(n \log n)$ [5, 定理 3.2]。为达到这一界限，维护当前与扫描线相交的射线的数据结构必须支持以下操作：以$\mathcal{O}(\log n)$ 的时间执行插入、删除和前驱搜索，并以$\mathcal{O}(1)$ 的时间访问左侧和右侧邻居。[5]

为了支持所需的操作，我们使用一个射线双向链表，并在其上层构建一棵 AVL 搜索树 [1]。图 4 展示了该数据结构的示意图。由于插入时已知射线沿扫描线的顺序——见算法 1::12——因此可以省去传统 AVL 数据结构中耗时$\mathcal{O}(\log n)$ 的搜索阶段，从而使插入操作（包括重新平衡）能够以$\mathcal{O}(1)$ 的摊销时间完成。不过，这项优化要求树中使用父指针。由于每次总是同时向扫描线数据结构中插入两条相邻射线，我们针对这种情况实现了一种特殊的插入操作，只需对两条射线执行一次重新平衡。对于删除操作，同样已知射线在扫描线中的位置——参见算法 1::30。因此，与插入操作类似，删除时无需搜索阶段，操作可在$\mathcal{O}(1)$ 的摊销时间内完成。该算法总是删除两条相邻射线 B_L and B_R and and，并将它们替换为 B^*。 B^* 与 B_L 的左邻居相同，且与 B^R 的右邻居相同。因此，我们只需在数据结构中用 B^* 替换 B_L，再删除 B^R，仅需执行一次重新平衡操作。

通过查找当前与扫描线相交且右侧有 p 的第一条射线 B_R，来确定点 p 的包围区域 – 算法 1::7 –。 B_R 的左邻居 B_L 必须在其左侧或位于其上。因此， B_L 和 B_R 包围 p，而 $B_L^r = B_R^l = R_q$ 给出 p 所在的区域。为便于查找，树中的每个内部节点都需要指向其子树中最右侧射线对应的叶节点。由于射线是复杂对象，我们使用指向相应叶节点的指针，以节省内存。

4.3 优先队列

算法开始时，优先队列 Q 会用所有输入点初始化。在事件处理过程中，交点事件和删除事件可能会被添加到 Q 或从中删除，因此需要使用可寻址优先队列。对象按照

[5] 即使邻居访问的时间复杂度为 $\mathcal{O}(\log n)$，$\mathcal{O}(n \log n)$ 的界仍然成立。

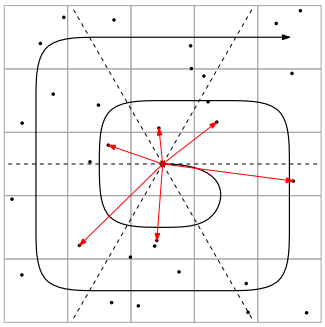


图 6 基于网格的 Yao 图构造算法。锥边界以虚线表示。算法按照粗线所示的顺序遍历网格单元。找到的 Yao 图边以红色标出。

公式 (1) 排序，因此键是（精确的）数值。我们的实验表明，通常对于 n 个输入点，在任意给定步骤中， Q 中只有大约 $\mathcal{O}(\sqrt{n})$ 个交点事件和删除事件。对所有事件使用同一个优先队列会导致动态优先队列操作开销很大。由于输入点事件在 Q 中是静态的，我们可以使用图 5 所示的两部分数据结构。输入点事件存储在一个数组中，按 Q 中的优先级排序，并用一个指针指向最小的未处理元素。交点事件和删除事件则存储在真正的可寻址优先队列中。我们为这部分数据结构使用可寻址二叉堆。top 操作需要比较优先队列中的最小元素与数组中指针所指向的元素，并返回两者中的较小值。弹出操作会相应地对优先队列执行常规弹出，或将数组指针移至下一个更大的元素。插入操作和删除操作可以直接访问优先队列，因为只有这部分数据结构是动态的。因此，实际的动态优先队列要小得多，从而使二叉堆中的下滤和上滤操作更加高效。较小的堆不仅降低了树的高度，也让堆更适合缓存。

这项优化或许也适用于其他算法：这类算法会将所有输入点初始化到优先队列中，并且在任意时刻，优先队列中动态添加的事件都只有少量。请注意，已处理的交点和删除事件的总 数远远超过输入点的数量，但同时位于优先队列中的此类事件却很少。

5 评估

本节将使用多个数据集，将我们的实现与其他 Yao 图生成算法进行比较评估。

5.1 竞争算法

如前所述，我们尚未发现任何 Chang 等人的 Yao 图算法实现。因此，我们将自己的实现与其他用于构造给定点集的 Yao 图的算法进行比较。我们的主要竞争对手是 CGAL 库的基于锥的伸展子图软件包 [12] 中的 Yao 图算法。由于我们尚未发现其他经过调优的 Yao 图构造实现，因此我们自行实现了另外两种算法作为对比。

首先，我们实现一种朴素方法 $\mathcal{O}(n^2)$算法，作为一个简单的基线。该算法比较所有点对之间的距离，并确定每个点在各个

k 锥中的最近邻。它只需要两种几何谓词：距离比较和有向直线侧测试。对于点对 p 和 q，可以先通过 $i = \sphericalangle pq/k$近似确定锥$\mathcal{C}_i^p$ 相对于 p 所在的位置 q 。然后，只需进行两次有向直线侧测试，即可精确确定 q 所在的锥。

其次，我们实现一种基于网格的算法。该算法将所有点放入均匀网格数据结构 [2], 中，该结构将所有输入点的包围盒划分为$\mathcal{O}(n)$个大小相等的单元。对于每个输入点 p，算法首先访问其自身所在的网格单元，并计算该单元内每个点 q 到 p 的距离，以及相对于 p而言 q 所在的锥。随后，算法以螺旋方式访问 p 所在单元周围的网格单元，参见图 6。对于每个访问到的单元，算法计算其中各点相对于 p 的距离和所在的锥，直到点 p 的所有锥都确定为止。如果在某个锥内找到了邻点，且相邻网格单元中没有任何点比 v更靠近 p，则该锥已确定。请注意，如果点 p的该锥内不存在其他输入点，那么在访问完所有网格单元之前，一些锥可能仍无法确定。尽管该算法的最坏情况时间复杂度仍为 $\mathcal{O}\left(n^2\right)$，但在实践中表现要好得多。

5.2 实验设置

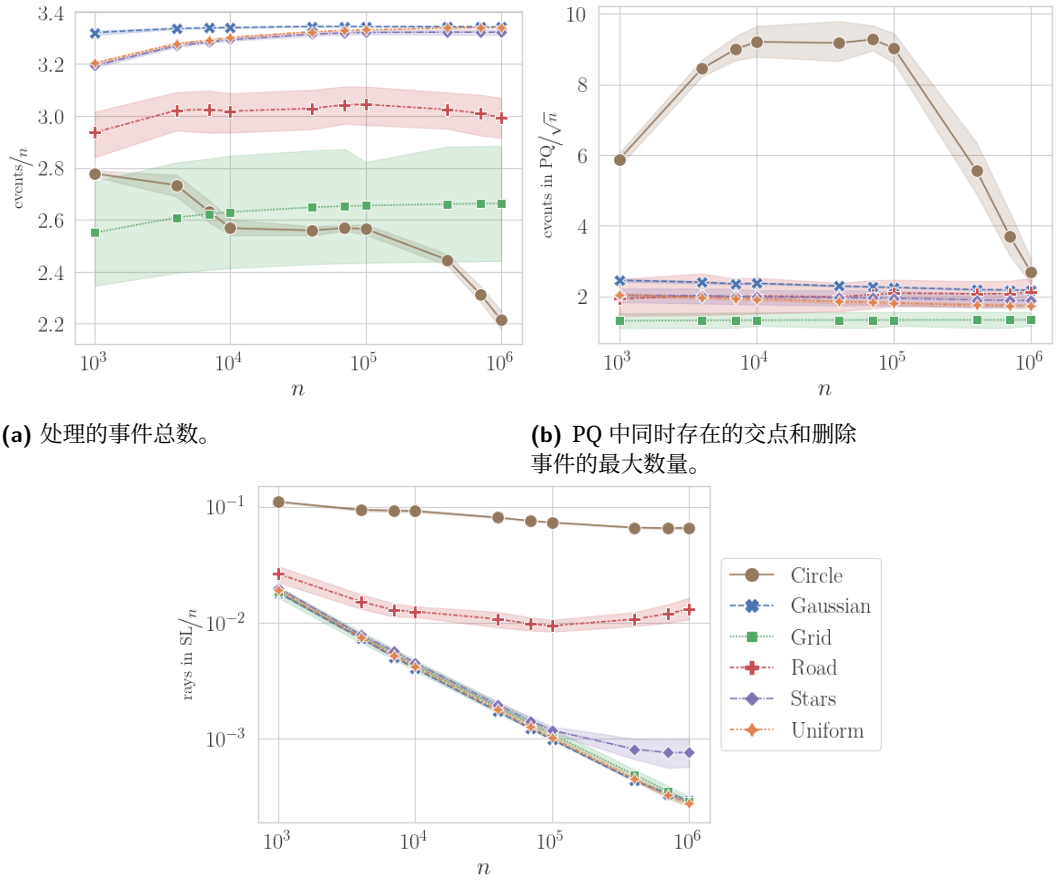
我们在各种合成数据集和真实世界数据集上测试所有算法。我们使用均匀分布和正态分布于单位正方形中的输入点集，以及位于圆周上和网格 [9] 交点处的点——前者是网格算法的最坏情况输入，后者则会导致数值稳定性较差。此外，我们还使用两个真实世界数据集：道路网络交叉口和星表。对于道路网络，我们使用 9^{th} DIMACS 实现挑战赛 [7]中的图。为了从美国全境图中生成具有所需规模的道路网络 n，我们选取一个随机位置，并以该位置为中心扩展一个矩形区域，直至区域内至少包含 n 个节点。美国城市中有许多点位于网格上，因此对数值稳定性构成挑战。此外，我们还使用 Gaia DR2 星表 [6], ，其中包含约 13。 亿颗恒星的天体位置。我们采用与生成道路网络相似的方法来生成具有所需规模的子图：以一个随机起始位置为中心扩展一个立方体，直至其中包含所需数量的恒星。随后，我们将所有恒星投影到 xy 平面上，作为实验的二维输入。附录中的图 10 展示了输入数据集及其生成的 Yao 图示例。

我们在 C++ 框架 YaoGraph 中实现了所有算法，该框架可在 GitHub 上获取。[6]我们的代码使用 GCC 12.1.0 和 CGAL 5.0.2 版本编译。所有实验均在一台配备 32 核 Intel Xeon Gold 6314U 处理器和 512 GiB 内存的服务器上运行。对于所有输入点分布，我们使用了三个不同的随机种子；除非另有说明，所有计算均使用 $k = 6$ 。

5.3 算法指标

首先，我们讨论扫描线算法的相关性质。图 7a 显示了该算法针对每个输入点处理的事件数量。每个输入点对应一个输入点事件，平均会产生 2.3 个交点和/或删除事件；除网格分布和道路图外，该数量几乎不受输入规模和分布影响，方差很小。网格分布和道路图的方差都较大，具体取决于锥体边界是否与网格直线重合

[6] https://github.com/dfunke/YaoGraph



(a) 处理的事件总数。

(b) PQ 中同时存在的交点和删除事件的最大数量。

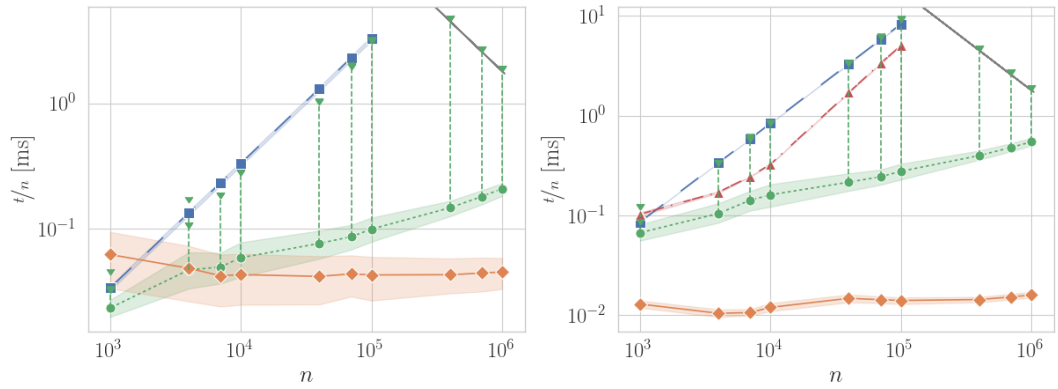
(c) 扫描线数据结构中的射线最大数量。

图 7 $k = 6$ 个锥的不同输入点分布和点集大小下的统计数据。误差带表示计算的不同锥之间的变化范围。

。图 7b 显示了算法执行期间，优先队列在任意时刻包含的交点和删除事件的最大数量。对于大多数研究的输入，该数量按 $\mathcal{O}(\sqrt{n})$ 的规模增长，这也促使我们选择如第 4.3 节所述的两部分优先队列。圆形分布的行为还需要进一步研究。算法执行期间扫描线数据结构在任意时刻包含的射线最大数量没有明显的增长规律，参见图 7c。对于大多数合成输入集，该数量按 $\mathcal{O}(\sqrt{n})$ 的规模增长；但对于圆形（≈10%）、道路（≈1 %）和星形（≈0.1 %）数据集，该数量接近输入规模的一个固定比例。

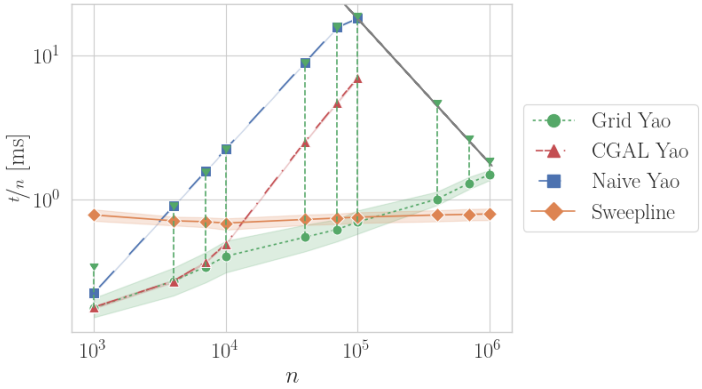
5.4 运行时间评估

图 8 展示了运行时间实验的结果。图 8a 至图 8c 展示了各算法的（缩放后）运行时间，并以误差带显示输入分布造成的差异。附录中的图 12 更详细地展示了不同分布下的运行时间。请注意，只有网格算法和扫描线算法会受到输入点分布的影响。如图 7a 所示，扫描线算法处理的事件数量在各种分布下都相对稳定。因此，该算法的运行时间变化很小。这也表明，扫描线数据结构的大小



(a) 非精确内核。

(b) CGAL EPIC 核。



(c) CGAL EPEC 核。

图 8 算法运行时间实验。使用 $k = 6$ 个锥，对不同输入规模进行了实验。误差带显示了不同输入点分布下的运行时间变化。对于网格算法，圆形分布使用三角形标记单独绘出。灰线表示每种算法 30 分钟的时间限制。改变锥的数量进行的实验使用 $n = 1 \times 10^5$ 个均匀分布的输入点。

对算法运行时间的影响可以忽略不计，因为在道路或圆形数据集上并未观察到更长的运行时间。与 CGAL 的高度优化核相比，我们的非精确内核的运行时间变化更大，这主要是由于网格分布中有许多点恰好位于锥边界上。扫描线算法明显优于 CGAL 的 Yao 图实现。此外，尽管扫描线算法需要进行复杂得多的计算，但对于非精确构造，它仍优于简单的网格算法。只有在精确构造的情况下，输入规模足够大时，才能抵消扫描线算法更昂贵的运算所带来的开销。与 EPIC 核相比，精确构造核会使运行时间增加 100 倍。不过，如果点恰好位于锥边界上，则必须进行精确构造才能得到正确结果，如附录中的图 10c 所示。对于网格算法，数据依赖性更为显著。该算法在大多数数据集上表现良好，但由于圆形内部有许多空网格单元，在圆形分布下会退化为朴素算法。

要计算具有 k 个锥的 Yao 图，扫描线算法需要 k 次遍历。这种线性关系可见于图 9。网格算法不依赖于 k ——唯一例外是点的邻域大小。不过，我们的实验表明，

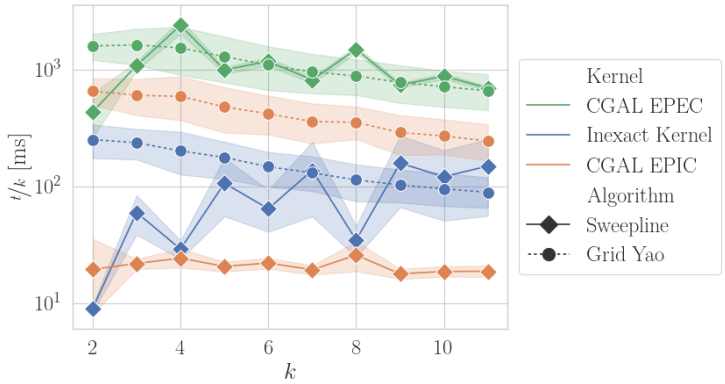


图 9 算法运行时间实验。锥数量变化的实验使用$n = 1 \times 10^5$ 均匀分布的输入点。

算法的运行时间会随着 k的增加而延长。我们认为，这是因为为了确定点 p的所有锥，需要遍历更多网格单元；锥越窄，某个锥中的点不在已遍历网格单元内的可能性就越大。由于运行时间过长，我们没有使用朴素算法或 CGAL 算法进行这些实验。CGAL 算法还要求每个锥遍历一次，而朴素算法的运行时间则不依赖于 k。

6 结论

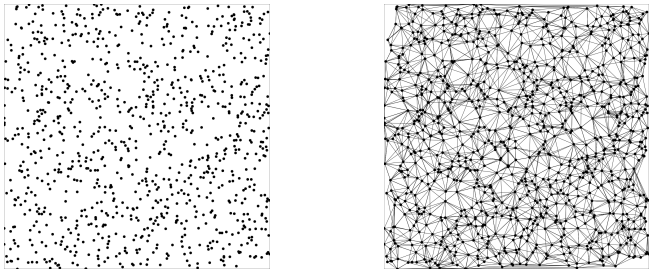
据我们所知，我们首次实现了 Chang 等人的最优$\mathcal{O}(n\log\ n)$ 时间 Yao 图算法。我们的实现对数据结构和算法操作进行了精心设计，因此性能至少比目前公开可用的 Yao 图实现快一个数量级，尤其是 CGAL 的基于锥的伸展子图软件包。此外，我们还介绍了一种非常简单的基于网格的 Yao 图算法，其性能也优于 CGAL 的实现，但对于较大的输入，其表现不及 Chang 等人的算法。不过，可以通过预先计算网格邻域到锥的映射来进一步改进该算法，从而只访问可能包含尚未处理的锥中点的网格单元。此外，该算法可轻易地按输入点进行并行化，而 Chang 等人的算法则只能轻易地按 k个锥进行并行化。扫描线遍历内部的并行化留待未来研究。

参考文献

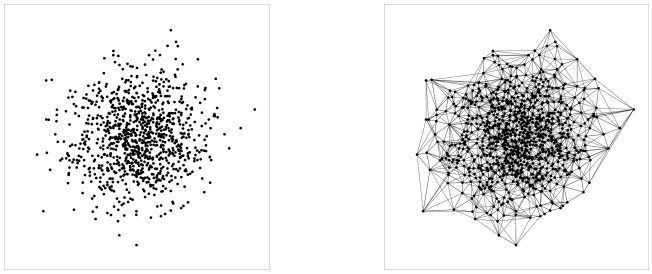
1 Georgy Maksimovich Adelson-Velsky 和 Evgeny Mikhailovich Landis。信息组织算法。载于《科学院学报》，第 146 卷，第 263–266 页。俄罗斯科学院，1962 年。2 V. Akman、W.R. Franklin、M. Kankanhalli 和 C. Narayanaswami。几何计算与均匀网格技术。《计算机辅助设计》，21(7)：410–420，1989 年。 doi:https://doi.org/10.1016/0010-4485(89)90125-5。3 J. L. Bentley 和 T. A. Ottmann。几何交点的报告与计数算法。《IEEE计算机汇刊》，第 643–647 页，1979 年。4 Hervé Brönnimann、Andreas Fabri、Geert-Jan Giezeman、Susan Hert、Michael Hoffmann、Lutz Kettner、Sylvain Pion 和 Stefan Schirra。二维和三维线性几何内核。载于《CGAL 用户与参考手册》。CGAL 编委会，第 5.5.1 版，2022 年。URL： https://doc.cgal.org/5.5.1/Manual/packages.html#PkgKernel23。5 Maw Shang Chang、Nen-Fu Huang 和 Chuan-Yi Tang。一种构造有向沃罗诺伊图和地理邻域图的最优算法。《信息处理快报》，35(5)：255–260，1990 年。URL: https://www.sciencedirect.com/science/article/pii/0020019090900542, doi:https://doi.org/10.1016/0020-0190(90)90054-2。6 Gaia 合作组。Gaia 数据发布 2：内容摘要与巡天属性。arXiv，(abs/1804.09365)，2018 年。7 Camil Demetrescu、Andrew V Goldberg 和 David S Johnson。最短路径问题：第九届 DIMACS 实现挑战赛，第 74 卷。美国数学学会，2009 年。8 Steven Fortune。一种用于沃罗诺伊图的扫描线算法。《算法学》，2(1)：153–174，1987 年。9 Daniel Funke、Peter Sanders 和 Vincent Winkler。并行德劳内三角剖分的负载均衡。载于 Ramin Yahyapour（编），Euro-Par 2019：并行处理，第 156–169 页，沙姆，2019 年。施普林格国际出版公司。10 Giri Narasimhan 和 Michiel Smid。《几何伸展网络》。剑桥大学出版社，2007 年。11 Christian Schindelhauer、Klaus Volbert 和 Martin Ziegler。应用于无线网络的几何伸展图。《计算几何》，36(3)：197–214，2007 年。12 Weisheng Si、Quincy Tse 和 Frédérik Paradis。基于锥的伸展子。载于《CGAL 用户与参考手册》。CGAL 编委会，第 5.4 版，2022 年。URL： https://doc.cgal.org/5.4/Manual/packages.html#PkgConeSpanners2。13 Andrew Chi-Chih Yao。关于在 k 维空间中构造最小生成树及相关问题。《SIAM 计算杂志》，11(4)：721–736，1982 年。 arXiv:https://doi.org/10.1137/0211059, doi:10.1137/0211059。14 Xiujuan Zhang、Jiguo Yu、Wei Li、Xiuzhen Cheng、Dongxiao Yu 和 Feng Zhao。信干噪比条件下无线网络中基于 Yao 图的伸展子构造局部算法。《IEEE/ACM 网络汇刊》，25(4)：2459–2472，2017 年。 doi:10.1109/TNET.2017.2688484。

A 评估

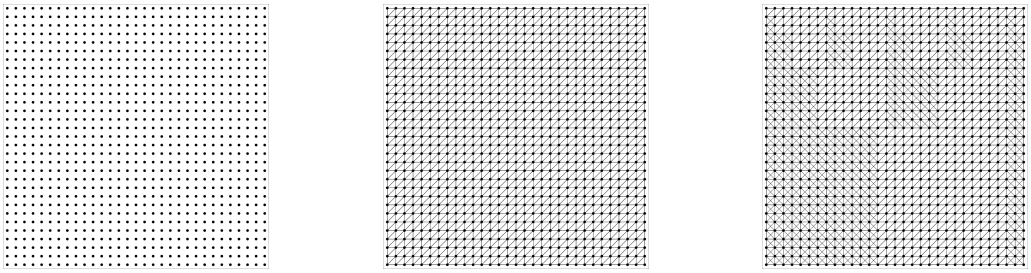
A.1 输入点分布



(a) 均匀分布。

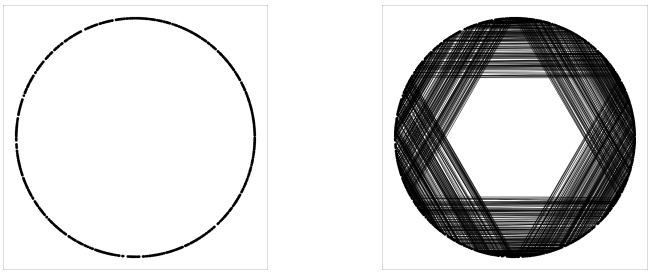


(b) 高斯分布。

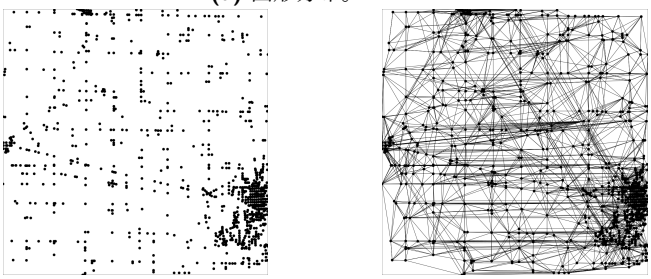


EPEC 核　　EPIC 核

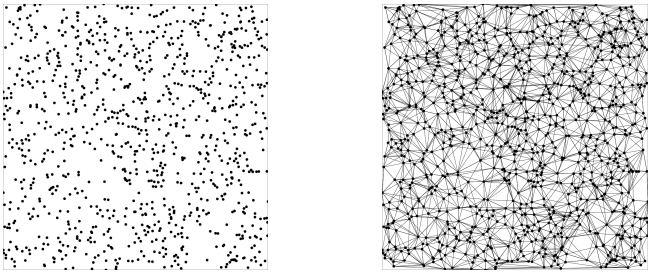
(c) 网格分布。



(d) 圆形分布。



(e) 道路数据集。



(f) 星形数据集。

图 10 $n = 1000$ 点的输入分布及其生成的 $k = 6$Yao 图。对于网格分布，展示了精确构造和非精确构造生成的图。

A.2 执行示例

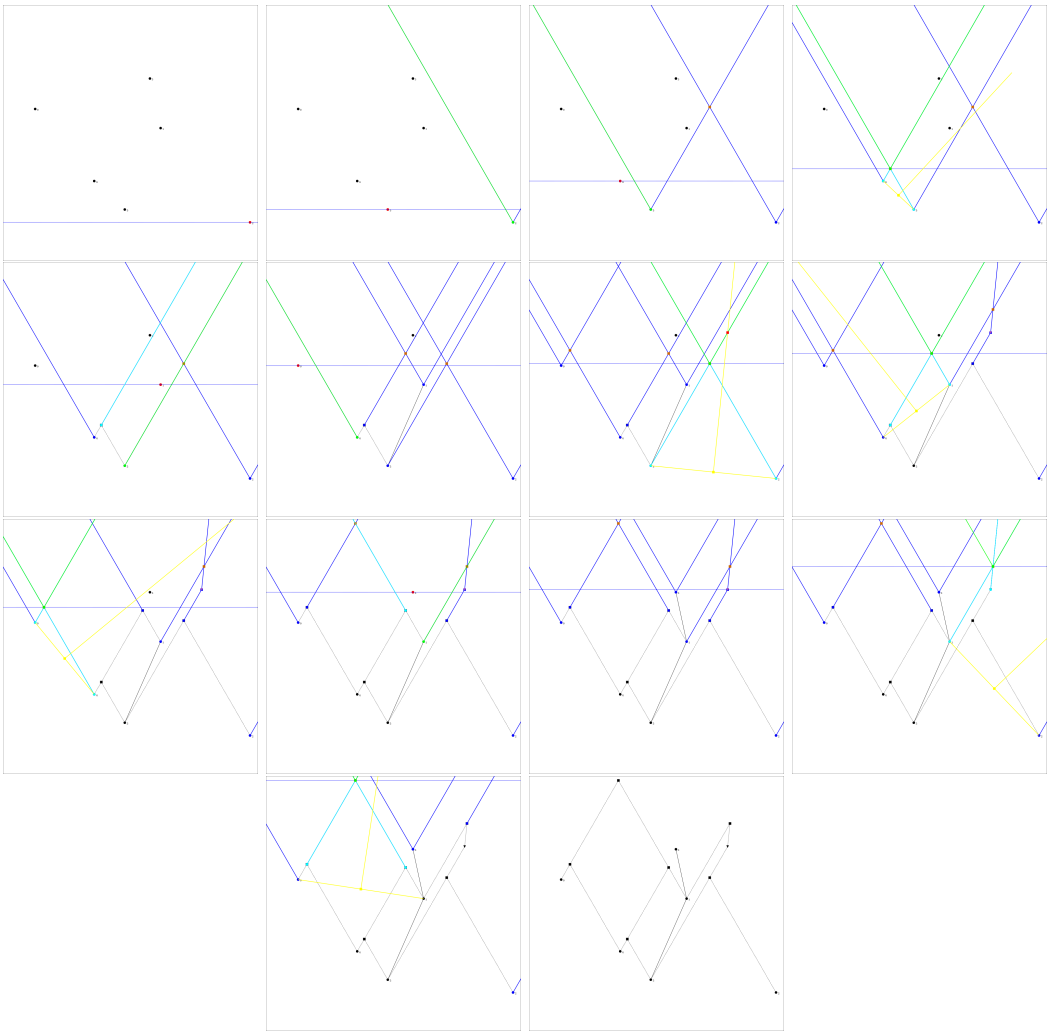
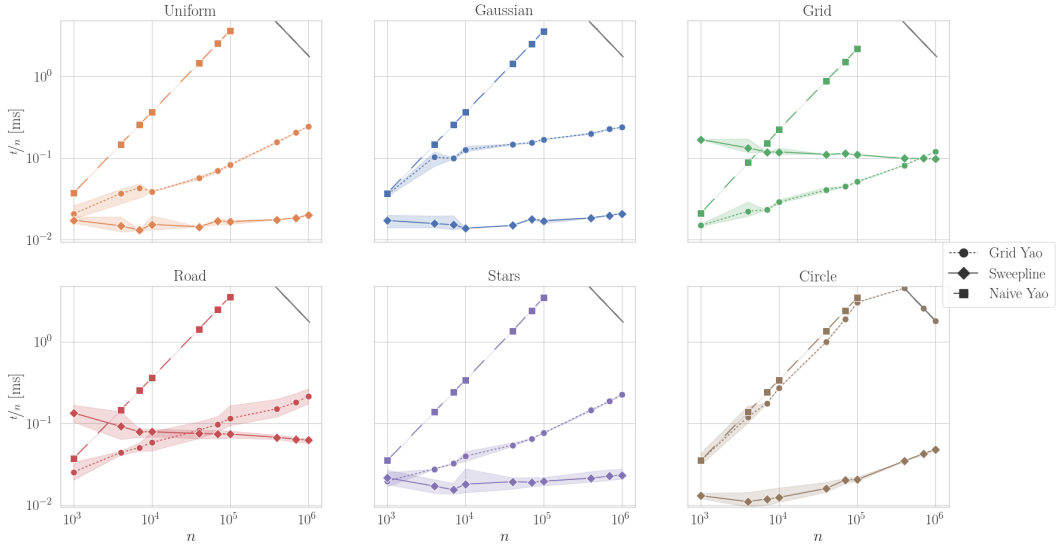
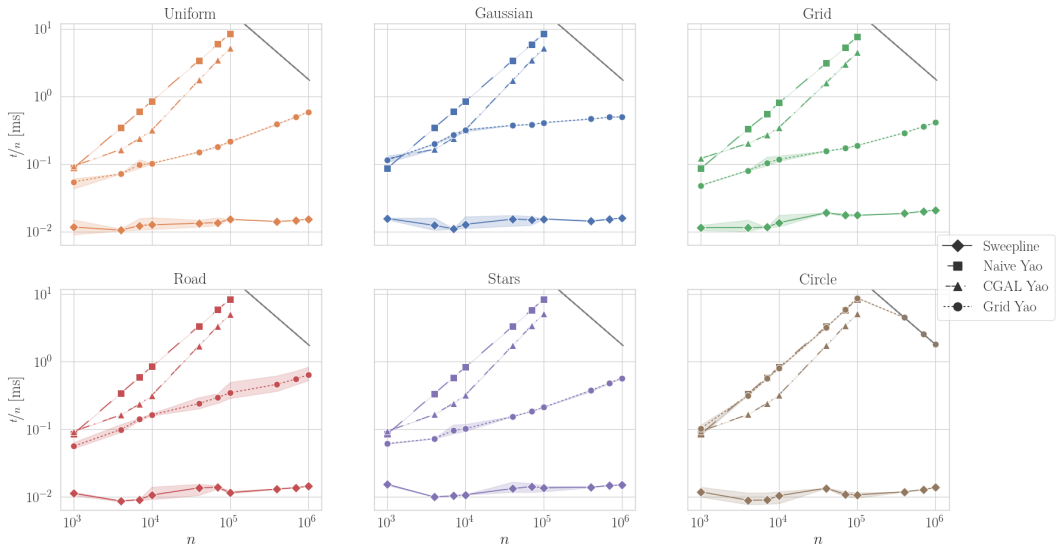


图 11 $n = 6$ 个点与锥 $\mathcal{C} = (4\pi/3, 5\pi/3)$ 的执行示例，结果如下（向上）。当前正在处理的点标为红色，扫描线为蓝色虚线。当前与扫描线相交的所有射线均为蓝色，左边界射线 B_L（青色）和右边界射线 B_R（绿色）除外。交点标为正方形，删除点标为三角形。对于相交事件，相交射线为青色，角平分线为黄色。Yao 图的边为黑色实线，已确定的锥边界为虚线。

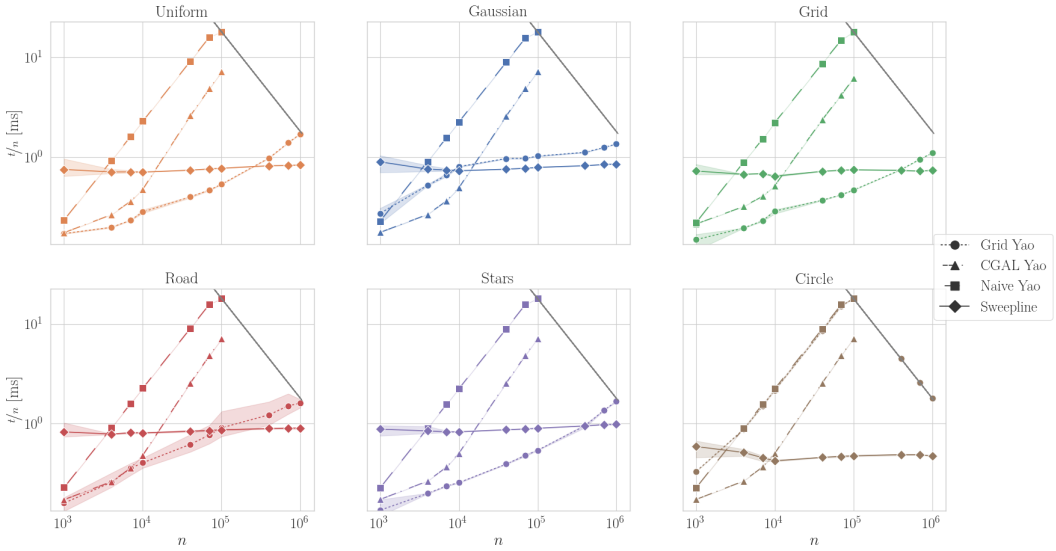
A.3 结果



(a) 非精确内核。



(b) CGAL EPIC 核。



(c) CGAL EPEC 核。

图 12 算法运行时间实验。使用 $k = 6$ 个锥，对不同输入规模进行了实验。灰色线表示每个算法 30 分钟的时间限制。