arXiv:2303.17204v1 [cs.DS] 30 Mar 2023

仙人掌图上的加权 k-中心问题的次二次时间算法

Binay Bhattacharya[1]、Sandip Das[2]和 Subhadeep Ranjan Dev[2]

[1]西蒙弗雷泽大学，本拿比，加拿大[2]印度统计研究所，加尔各答，印度

2023年3月31日

摘要

图中的加权 k-中心问题是一类经典的设施选址问题：我们在图上放置 k 个中心，使顶点到其最近中心的最大加权距离最小。我们研究底层图为具有 n 个顶点的仙人掌图时的这一问题，并给出一个时间复杂度为 $O(n \log^2 n)$ 的算法。该时间复杂度优于 Ben-Moshe 等人提出的 $O(n^2)$ 时间算法 [2], ，后者是目前最先进的算法。

1 引言

设施选址问题是运筹学中的一个经典问题，在图和几何领域都得到了广泛研究。在图中，顶点具有某种需求，而边具有相应的长度。边上的长度对应于其两个端点顶点之间的距离。其目标是在图的边和顶点上确定一定数量的位置，以便在这些位置放置设施。设施的布置应满足图中顶点的需求，同时使某个预先定义的目标函数最小化。

在加权 k-中心问题中，目标是放置 k 个设施点，使顶点与其最近设施点之间的最大加权距离最小。Kariv 和 Hakimi [12] 于 1979 年提出了这一问题，并证明了它对于一般图而言是 NP难问题。事实上，他们证明了一个更强的结论，表明该问题在最大度为 3 的平面图上也是 NP难问题。然而，研究表明，当图类受到限制或 k 存在某种上界时，该问题具有高效的多项式运行时间。例如，对于树，Wang 和 Zhang [13] 最近证明，加权k-中心问题可以在 $O(n \log n)$ 时间内求解，其中 n 是该树的顶点数。如果 k 是常数，Bhattacharya 等人 [3] 提出了一个

在 $O(n)$ 时间内求解该树上问题的最优解。对于仙人掌图，目前最先进的是 Ben-Moshe 等人提出的一个 $O(n^2)$ 时间算法 [2]。当 k 被限制为 1 或 2 时，仙人掌图上的运行时间分别降至 $O(n\log n)$ 和 $O(n\log^3 n)$ 时间 [4]。对于部分 p-树，Bhattacharya 和 Shi [5] 证明，当设施点仅限于放置在顶点上时，加权 k-中心问题可以在 $O(n^k \log^{2p} n)$ 时间内求解。

Wang 和 Zhang [13] 针对树上的加权 k-中心问题提出的算法使用了 Frederickson 的参数搜索技术 [7]，将运行时间从 $O(n \log n + k \log^2 n \log(n/k))$ [1] 降至 $O(n\log n)$。我们将 Frederickson 的技术推广开来，提出了一种运行时间为 $O(n\log^2 n)$ 的算法来解决该问题。这是首次将此类推广应用于树以外的图类，也是该问题的首个次二次时间算法。可行性测试是该问题的判定版本：给定一个代价 λ，测试返回最优代价（未知）是否不超过 λ。此外，针对加权 k-中心问题，我们还提出了一种可行性测试算法，其运行时间为最优的 $O(n)$。

本文其余部分的组织如下。第 2 节给出加权 k-中心问题的形式化定义以及仙人掌图的树表示。我们介绍一些关于圆弧图的结果，并将在后续各节中使用这些结果。第 3 节介绍我们的可行性测试算法。对于任意输入 λ，该测试确定 $\lambda \geq \lambda^*$，其中 λ^* 是加权 k-中心问题的最优代价。第 4 节证明线性排列的一些性质，并将在第 5 节中使用相关结果。第 5 节定义仙人掌茎，它本质上是仙人掌图的一个子图，也是 Frederickson [7]所定义的（路径）茎的推广。随后，我们提出一种运行时间为 $O(n\log^2 n)$ 的算法，用于求解仙人茎上的加权 k-中心问题。我们还展示了另一种技术，利用该技术可以得到运行时间为 $O(n \log n + kn \log n)$ 的算法。第 6 节介绍我们针对仙人掌图的 $O(n\log^2 n + kn \log n)$ 时间算法。最后，我们在第 7 节作出总结。

2 预备知识

2.1 问题定义

设 G 是一个图，其顶点集为 $V(G)$，边集为 $E(G)$，且 $|V(G)| = n$。如果 G 中任意两个环不共享公共边，则称其为仙人掌图。每个顶点$v \in V(G)$都关联一个非负权重 $w(v)$，每条边 $e \in E(G)$ 都关联一个正长度 $l(e)$。我们将 e 解释为长度为 $l(e)$ 的线段，从而可以指代 e 上的任意点 x。G 的所有边上的全部点构成的集合记为 $A(G)$。对于任意两个点 x、$y \in A(G)$，x 与 y 之间的距离记为 $d_G(x, y)$，其定义为 G 中连接它们的最短路径的长度。如果下标 G 显然，则省略不写。对于一个点 $x \in A(G)$ 和一个顶点 $v \in V(G)$，点 x 与顶点v 之间的加权距离等于 $w(v) \cdot d(x, v)$。给定一个点集 X，用 X 覆盖 G 的顶点所需的代价表示为

$$\max_{v\in V(G)}\left\{\min_{x\in X}\{w(v)\cdot d(x,v)\}\right\}$$

加权 k-中心问题的目标是在 G 中找到一个包含 k 个点的集合 X，使覆盖 V（G）的代价最小。该代价也称为 G 中加权 k-中心问题的最优代价。

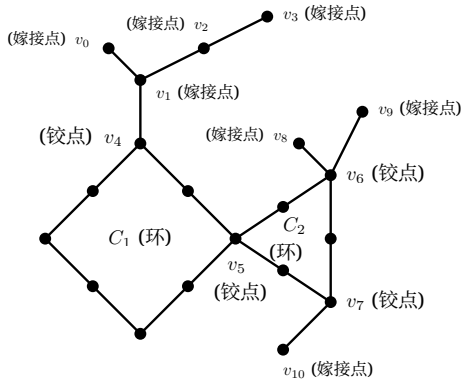
我们称点（或中心） $x \in A(G)$以代价 λ覆盖顶点 $v \in V(G)$，当且仅当 $w(v) \cdot d(x, v) \leq \lambda$。扩展此定义，我们称中心集合 X 以代价 λ 覆盖$V(G)$，当对于所有顶点 $v \in V(G)$，其在 X 中的最近中心以代价 λ覆盖 v。如果 λ 已被充分理解，我们有时不会明确提及它。

2.2 仙人掌图的树表示

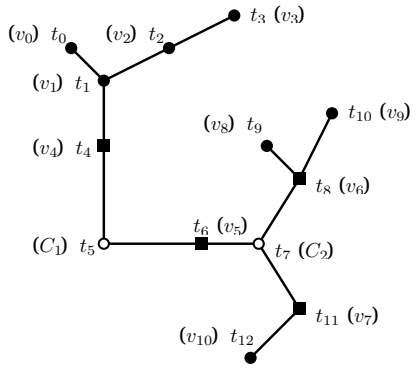
给定仙人掌图 G， G 中的顶点如果不属于 G中的任何环，则称为嫁接点。如果它属于 G 中的某个环且度至少为3，则称为铰点。其余顶点（如果存在）均恰好属于G中的一个环。

我们将 G 转换为一个新图 T，其中 T、 $V(T)$的节点表示 G中的嫁接点、铰点或环。如果节点 $t_i \in V(T)$表示 G中的一个环，则向表示该环的铰点的每个节点 $t_j \in V(T)$添加一条边。如果 t_i， $t_j \in V(T)$均不表示环，则当且仅当它们所表示的 G 中的顶点相邻时，它们相邻。

请注意， $V(T)$ 中分别表示 G 的一个嫁接点和一个环的两个节点永远不会相邻。由于 G 是仙人掌图，因此由此形成的图 T 是一棵树。我们将 T 称为 G [2] 的 树表示。图 1 展示了一个示例。



a 一个仙人掌图 G。



b 以 t_0 为根的 G 的树表示 T。对于所有 $t \in V(T)$， t 所表示的顶点位于方括号内。

图 1： t_0 的关联顶点是 v_0，而 t_4 和 t_5 的关联顶点都是 v_4。

设 $t_r \in V(T)$ 为任意一个叶节点，并将其指定为 T 的根。这就在 $V(T)$ 的节点上建立了父子关系。现在我们将

每个节点 $t \in V(T)$ 映射到一个顶点 $v \in V(G)$。我们称 t 的关联顶点为 v。

对于每个表示嫁接点或铰点 $v_i \in V(G)$ 的 $t_i \in V(T)$，我们将 v_i 指定为 t_i 的关联顶点。对于每个表示 G 的一个环的 $t_j \in V(T)$，令 t'_j 为其父节点。由于 t_j 表示一个环，t'_j 始终表示 G 的一个铰点。t_j 的关联顶点设为 t'_j 的关联顶点。如果 t_j 表示 G 的一个环，且同时是根（$t_j = t_r$），我们任意选取该环的一个顶点，并将其设为 t_j 的关联顶点。我们将 t_r 的关联顶点（记作 v_r）称为 G 的根。

引理 1 (Ben-Moshe 等人 [2])。给定一个具有 $|V(G)| = n$ 的仙人掌图 G，其树表示 T 可以在 $O(n)$ 时间内求出。

2.3 圆弧

圆弧 是圆上两个不同点之间的一段弧。本节介绍对一组圆弧进行的一些操作。设 C 为一组包含 m 条圆弧的集合。我们假设 C 中所有弧的中点互不相同，并且按顺时针方向排列。令 c_1、c_2、.....、c_m 为 C 的给定排序顺序。对于 $c_i \in C$、$i < m$，c_{i+1} 称为 C 的顺时针邻居 。对于 c_m，c_0 是其顺时针邻居。逆时针邻弧 的定义类似。

如果存在另一条弧 $c' \in C$ 被弧 $c \in C$ 完全包含，即 $c \supset c'$，则称弧 $c \in C$ 为超弧 。现在证明如下结论。

引理 2. 可以在 $O(m)$ 时间内移除 C 中的所有超弧。

证明。将 C 中的所有弧标记为未访问。任意选取一对相邻的弧 a，$b \in C$。注意，b 是 a的顺时针邻居。当 a 尚未标记为已访问时，重复以下步骤。

如果 a 被 b包含，则将 b 替换为其顺时针邻居，并从 C中移除旧的 b。否则，将 a 标记为已访问，并分别将 a 和 b 更新为各自的顺时针邻居。

再沿逆时针方向将整个过程重复一次。

上述步骤会移除 C中的所有超弧。由于 C 中的所有弧都分别沿顺时针方向和逆时针方向遍历一次，因此该算法需要 $O(m)$ 时间。□ □

引理 2 的算法伪代码如算法 1 所示。

如果一组圆弧不包含任何超弧，则称其为真。在本节余下部分，我们将 C 视为一组真圆弧。C 的最小刺穿集 χ 是一个基数最小的点集，使得 C 中的每条弧都包含 χ中的某个点。χ 的基数称为 C的刺穿数。Hsu 等人 [11] 证明，一组 m 圆弧的最大独立集可以在 $O(m)$时间内找到。利用相同的技术可以证明以下结果。

引理 3 (Hsu 等人 [11])。*</style}*当 C 中弧的中点按排序顺序给出时，可以在线性时间内求出 C 的穿刺集。

算法 1 移除超弧

将 C 中的所有弧标记为未访问
令 $a \in C$
将 $b \leftarrow$ 设为 a 的顺时针邻居
当 a 尚未标记为已访问 **时**
    如果 $a \subset b$，**则**
        将 $b' \leftarrow$ 设置为 b 的顺时针邻居
        从 C 中移除 b
        设置 $b \leftarrow b'$
    else
        将 a 标记为已访问
        设置 $a \leftarrow b$
        将 $b \leftarrow$ 设置为 b 的顺时针邻居
    结束 if
结束 while
沿逆时针方向再完整重复一次整个过程

每条弧 $c \in C$ 都有两个端点。从 c 的内部出发，在顺时针遍历中首先遇到的端点是 c 的顺时针端点 。另一个端点是 c 的逆时针端点 。 c 的顺时针后继是弧 c' ：从 c 的顺时针端点出发顺时针遍历时首先遇到，且不与 c 相交的弧。类似地，我们定义 c 的逆时针下一个 。请参见图2。

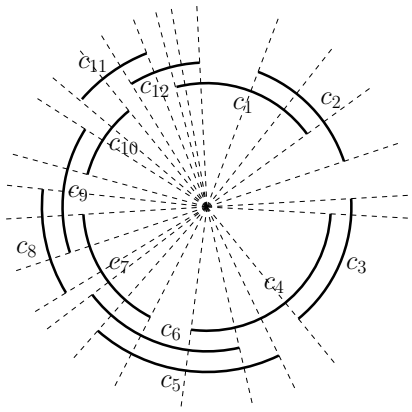


图2：对于圆弧 c_1，其顺时针邻居是 c_2，顺时针后继是 c_3 ；例如，对于 $q = 3$ ，其顺时针 q-后继是 c_8。

由于 C 的端点已按排序顺序给出，我们提出如下论断。

引理 4。 C 中所有弧的顺时针（逆时针）后继可以在

在 $O(m)$ 时间内找到。

证明。将 C 中的所有弧标记为已访问。任意选择一条弧 $a \in C$，并设置$b \leftarrow a$。当 a 未被标记为已访问时，重复以下步骤。

如果 a 与 b 相交，则将 b 更新为其顺时针邻居。否则，将 a 标记为已访问，并将 b 设为 a的顺时针邻居。将 a 更新为其顺时针邻居。

上述步骤在 $O(m)$ 时间内为 C 中的所有弧设置顺时针后继。所有弧的逆时针下一个可以用类似方法找到。 □ □

引理 4 中算法的伪代码见算法 2。寻找 C 中所有弧的逆时针下一个的算法与此类似。

算法 2 顺时针后继

```
将 C 中的所有弧标记为未访问
令 a ∈ C
将 b ← a设为
当 a 尚未标记为已访问时，执行
    如果 a ∩ b ≠ ∅，则
        将 b ← 设为 b的顺时针邻居
    else
        将 b 设为 a的顺时针后继
        将 a 标记为已访问
        将 a ← 设为 a的顺时针邻居
    结束 if
结束 while
```

对于任意正整数 q，令 $[q]$ 表示集合 $\{1, 2, \ldots, q\}$。现在定义弧 $c_0 \in C$的顺时针 q后继。令 c_0，c_1，……，c_q 为 C 的一列圆弧，使得对于任意 $i \in [q]$，c_i 是 c_{i-1}的顺时针后继。弧 c_q称为 c_0的顺时针 q阶后继。注意，对于任意 $i \in [q]$，弧 c_i 是 c_0的顺时针 i阶后继。逆时针 q阶后继的定义类似。参见图 2。

引理5。 C 中所有弧的顺时针（逆时针） q阶后继可在 $O(m)$ 时间内找到。

证明。引理5的伪代码如算法3所示。该算法使用Gavruskin等人的区间相容性森林数据结构 [10]，通过将其扩展到圆弧并为其适当命名，将其称为相容性图数据结构。

令 G_C 表示 C 的顺时针相容性图，该图的顶点分别对应 C 的每条弧，并且当 b 是 a的顺时针后继时，存在一条从 a 到 b的边 a，$b \in C$。图 G_C 可能是不连通的，其中每个连通分量恰好包含一个有向环。注意，该分量中不存在其他无向环。

仔细实现算法3将得到一个 $O(m)$时间的算法。逆时针情形的算法类似。 □ □

算法3 q顺时针后继

将 C 的所有弧初始化为未访问
对 G 的每个连通分量 G' **执行**
    设 r 为 G' 有向环上的一条弧
    移除连接 r 与其顺时针后继的边，将 G' 转换为一棵以 r 为根的树 T'
    设置 $a \leftarrow r$
    初始化一个大小为 q 的双端队列 Q，使得 Q 中的第 i^{th} 个元素是 a 的顺时针 i-下一个
    当 r 未标记为已访问时 **执行**
        如果 a 有一个未访问的子节点 b **则**
            更新 Q：将 a 插入为其第一个元素，并删除其最后一个元素
            设定 $a \leftarrow b$
        else
            令 b 为 Q 中的最后一个元素
            将 b 设为 a 的顺时针 q 后继，并将 a 标记为已访问
            更新 Q，移除其第一个元素，并插入顺时针后继 b 作为其最后一个元素
            设置 $a \leftarrow$ 的父节点为 a
        结束 if
    结束 while
结束 for

3 可行性测试

在本节中，我们提出一种算法，用于在与仙人掌图规模呈线性关系的时间内求解加权 k-中心问题的判定版本。文献中通常将该问题称为可行性测试 。给定仙人掌网络 G，设 $\lambda^* \in \mathbb{R}_{\geq 0}$ 为 G中加权 k-中心问题的最优代价。对于输入 $\lambda \in \mathbb{R}_{\geq 0}$，可行性测试要求判断 $\lambda \geq \lambda^*$是否成立。注意，最优代价 λ^* 是一个未知值。如果 $\lambda \geq \lambda^*$，则称 λ 是可行的；否则， $\lambda < \lambda^*$，并称 λ 是不可行的。

对于给定的 λ，我们的算法通过放置数量最少的中心X来运行，使得 G 中的每个顶点都能以代价 λ被 X 中的某个中心覆盖。如果 $|X| \leq k$，则 λ 是可行的；否则 $|X| > k$，而 λ 是不可行的。该算法使用 G 的树表示 T，以便确定 G的顶点处理顺序。回想一下， T 以节点 t_r为根。我们按照自底向上顺序处理 T 的节点，先访问叶节点，然后访问其子节点已处理完毕的节点。

对于 $t \in V(T)$，令 T_t 表示以 t 为根的 T子树，并令 G_t 表示 G 所代表的子图。对于每个节点 $t \in V(T)$，我们存储两个值 close(t) 和 cover(t)，定义如下。

考虑可行性测试算法的一个中间步骤，此时 G_t 中的所有顶点都已处理。算法可能已在其中放置了

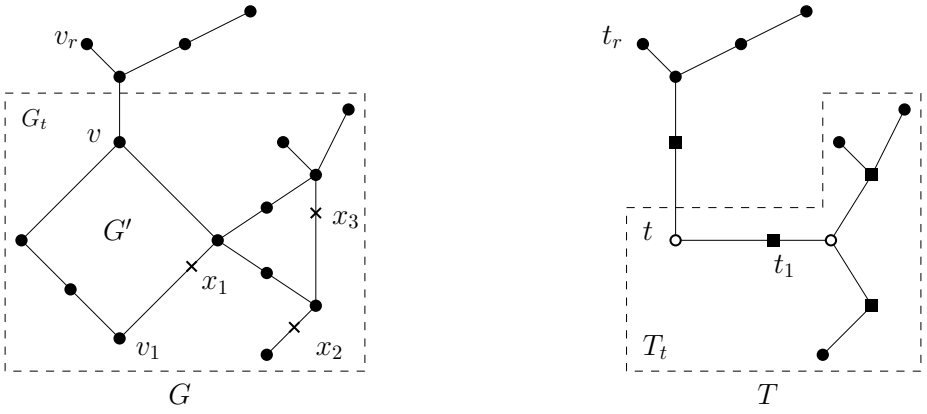


图3：子图 T_t 和 G_t。 $V(G)$ 中的顶点 v 和 v_1 分别是 $V(T)$ 中 t 和 t_1 的关联顶点。节点 t_1 是 t 的子节点， $t \in V(T)$ 表示 G 的环 H ，以及 $X_t = \{x_1, x_2, x_3\}$。

一些中心。令这些中心构成的集合表示为 X_t。 G_t 的未覆盖顶点是指那些无法以 λ 的代价被 X_t 覆盖的顶点。

令 $v_t \in V(G)$ 为 $t \in V(T)$ 的关联顶点。函数 cover(t) 表示在 G_t 之外放置一个中心，使其距 t 的距离最大，同时以代价 λ 覆盖 G_t 中所有未覆盖顶点时的最大距离。令 $V' \subseteq V(G_t)$ 为 G_t 中未覆盖顶点的集合。我们将 cover(t) 定义如下。

$$\mathrm{cover}(t) = \begin{cases} \min\{\lambda / w(v) - d(v, v_t) \mid \forall v \in V'\} & \text{if } V' \neq \varnothing \\ \infty & \text{otherwise} \end{cases}$$

函数 close(t) 存储 G_t 内最近中心距 v_t 的距离。我们将 close(v) 定义如下。

$$\mathrm{close}(t) = \begin{cases} \min\{d(x, v_t) \mid \forall x \in X_t\} & \text{if } X_t \neq \varnothing \\ \infty & \text{otherwise} \end{cases}$$

回顾一下，我们按照自底向上的顺序遍历 T 的节点。我们还维护一个变量 $count$，用于统计截至目前放置的中心数量。令 t 为当前所在的 T 中的节点，并令 v 为其关联顶点。在第 3.1 节和第 3.2 节中，我们将说明如何在 G_t 内放置中心，具体取决于 t 表示 G 的铰点 / 嫁接点还是环。

3.1 t 表示 G的铰点或嫁接点

令 $V_t \subseteq V(T)$ 为 t 的子节点集合。对于每个 $t_i \in V_t$，令 v_i 为其对应的顶点。

令 $V_1 \subseteq V_t$，使得 $\forall t_i \in V_1$ 覆盖 t_i 的 cover($< d$)(v_i， v_t)。注意，对于所有这样的 $t_i \in V_1$，为了覆盖 G_{t_i} 中其余未覆盖的顶点，必须在边 (v_i， v_t) 上放置一个中心。我们通过在边 (v_t， v_i) 上放置一个中心来实现，令其与 v_i 的距离为 cover(t_i)，并将 $count$ 加一。由于 T_{t_i} 中的所有顶点现在都已覆盖，我们将 cover(t_i) 设为 ∞。

根据定义，我们现在将 close(t) 设为放置在 G_t 中的所有中心到 v_t 的距离中的最小值。令 x 为这样的一个中心，即 x 是距离 v_t 最近的中心。如果 G_t 中没有放置任何中心，则将 close(t) 设为 ∞。

现在，令 $V_2 \subseteq V_t \setminus V_1$ 满足 $\forall t_j \in V_2$，cover(t_j) $\geq d(v_t, v_j)$ + close(t)。注意，对于所有这样的 $t_j \in V_2$， G_{t_j} 中的所有顶点要么由放置在 G_{t_j} 内部的中心覆盖，要么由中心 x 覆盖（如果该中心存在）。因此，我们将 cover(t_j) 设为 ∞。

最后，我们更新 cover(t)。我们将参数 δ 定义为

$$\delta = \min\{\text{cover}(t_i) - d(v_t, v_i) \mid \text{for all child } t_i \text{ of } t\}$$

如果 close(t) $\leq \lambda / w(v_t)$，则将 cover(t) $= \delta$；否则，cover(t) $= \min\{\lambda / w(v_t)\ \delta\}$。

3.2 t 表示一个由 G构成的环

令 H 为 G 的环，且由 t 表示，并令 $m = |V(H)|$。对于顶点 $v_j \in V(H)$，令 near(v_j) 表示 G_t 中最近中心到 v_j 的距离。near(·) 的定义与 close(·) 的定义类似。close(·) 定义在 $V(T)$ 的节点上，而 near(·) 则定义在 G 中各个环的顶点上。注意，计算 near(v_j) $\forall v_j \in V(H)$ 所需的唯一信息是 close(t_i) $\forall t_i \in V_t$。因此，不难证明， $\forall v_j \in V(H)$，near(v_j) 可以在 $O(m)$ 时间内计算出来。

如果 $v_j \neq v_t$，那么当 near(v_j) $>$ cover(t_j) 时， v_j 未覆盖，其中 $t_j \in V(T)$ 表示 v_j。如果 $v_j = v_t$ 或者不是铰点，那么当 near(v_j) $>\lambda/w(v_j)$ 时， v_j 未覆盖。我们的目标是通过在 H 中放置零个或多个中心，并在 G_t 外最多放置一个中心，覆盖 H 中所有未覆盖顶点。

我们将这个在 H 中放置中心的问题，转换为等价的、针对一组圆弧 C 的刺穿集问题，方法如下。对于 $i \in [m]$，如果 v_i 是一个铰点且如果$v_i \neq v_t$，令 $t_i \in V(T)$ 为表示 v_i 的节点。我们令 $l_i = 2 \times$ 覆盖t_i。否则，我们令 $l_i = 2\lambda/w(v_i)$。我们在 C 中插入一条长度为 l_i、以 v_i为中心的圆弧。

我们的目标是放置一组刺穿点，使所放置的中心总数最少。令 $C' \subseteq C$ 为不与 v_t 相交的圆弧子集。令 q 和 q' 分别为 C和 C'的最小刺穿集的大小。注意， $q - 1 \leq q' \leq q$。如果 $q' = q - 1$，那么与其将一个中心 x 放置在 v_t，不如在允许距离内将其移至 G_t之外。

因此，我们找到了一个刺点集合 χ，使得满足以下任一情况：（a） H 的所有未覆盖顶点都由 χ覆盖，并且将 $\exists x \in \chi$ 放置在尽可能靠近 v_t的位置；（b） H 的未覆盖顶点并非全部由 χ 覆盖，还需要再设置一个中心

需要将 x 放置在 G_t 之外以覆盖它们。在后一种情况下，选择 χ，使得 x 能够放置在尽可能远离 v_t的位置。如果 $q' = q$，则情况（a）成立；否则 $q' = q - 1$，情况（b）成立。给定按循环顺序排列的 H，我们展示如何在 $O(m)$ 时间内计算集合 χ。

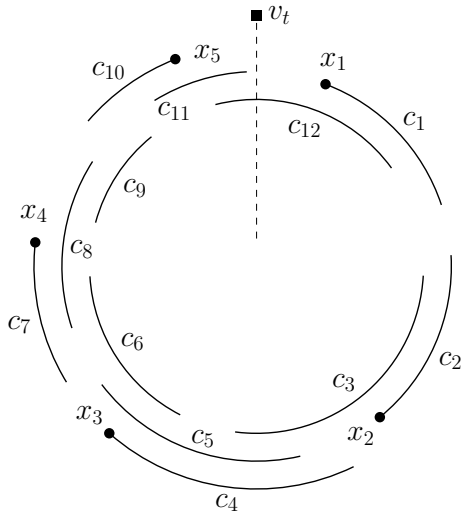


图4：用于 $q/q' = 5$ 的集合 $\chi(c_1)$。

3.2.1 情况（a）

在这种情况下，我们需要找到 C 的最小穿刺集 χ，使得其中一个刺点 $x \in \chi$ 距离 v_t 最近。我们检查 χ 的所有可能配置，并选取其中包含距离 v_t 最近的点的配置。令 c_1 为从 v_t 沿顺时针方向遇到的第一个与其不重叠的弧。令 c_j 为 c_1 的顺时针后继。对于每个 $c_i \in \{c_1, c_2, \ldots, c_{j-1}\}$，我们按如下方式构造刺点集合 $\chi(c_j)$。

我们将第一个刺点 x_1 放在 c_i 的逆时针端点处。接下来，将刺点 x_2 放在 c_i 的顺时针邻居（设为 $c_{i+i'_1}$）的顺时针端点处。然后，将刺点 x_3 放在 $c_{i+i'_1}$ 的顺时针后继（设为 $c_{i+i'_2}$）的顺时针端点处。我们以这种方式继续放置点，并将第 q 个刺点 x_q 放在 c_i 的顺时针第 $q-1$ 个后继（设为 $c_{i+i'_{q-1}}$）的顺时针端点处。如果 $c_{i+i'_{q-1}}$ 的顺时针后继与 x_1 重叠，则集合 $\chi(c_i)$ 是 C 的一个有效最小穿刺集。

注意，对于任意 $c_i \in \{c_1, c_2, \ldots, c_{j-1}\}$，$\chi(c_i)$ 中距离 v_t 最近的点要么是 x_1，要么是 y_q。因此，我们只需为每个 $\chi(c_i)$ 计算第一个和第q个刺点即可，这可以在 $O(1)$时间内完成。我们选择χ 作为一个 $\chi(c_i)$，它是有效刺穿集，且其中有一个点距离 v_t最近。我们计算 χ 中的每个点，并在 H 中对应的位置放置一个中心。我们将 $count$ 增加 q，并相应地更新 close(t) 和 cover(t)。

3.2.2 情况（b）

在这种情况下，我们在 H上找到一组 q' 个中心，使得位于G_t之外的中心——它覆盖了 G_t 到 v_t之间剩余的未覆盖顶点——尽可能远离 v_t。对于每个 $c_i \in \{c_1, c_2, \ldots, c_{j-1}\}$，我们按照前文所述构造一组刺点 χ（c_j），其中用 q'替换 q。

如前所述，对于任意 $c_i \in \{c_1, c_2, \ldots, c_{j-1}\}$，$\chi(c_i)$ 上距离 v_t 最近的点要么是 x_1，要么是 $x_{q'}$。因此，只需为每个 $\chi(c_i)$ 计算第一个和第 q' 个刺穿点即可，这可以在 $O(1)$ 时间内完成。

令 $^{c_{i-i'_0}}$ 为 c_i 的逆时针下一个，$^{c_{i+i'_{q'}}}$ 为 $^{c_{i+i'_{q'-1}}}$ 的顺时针后继，并令 v_1 和 v_2 分别为 H 中由它们表示的顶点。令 V' 为 G_t 中的中心集合：放置与 $\chi(c_i)$ 对应的中心后，这些中心仍未被覆盖。对于从外部覆盖 G_t 中其余未覆盖顶点的中心，距离 v_t 最远的中心的距离为 $\min\{\alpha(v_1) - d(v_1, v_t)\ \alpha(v_2) - d(v_2, v_t)\}$，其中 $\alpha(v_l)$ $(l = \{1, 2\})$ 在 v_l 为铰点时等于 $\text{cover}(v_l)$，否则等于 λ/v_l。我们将此值记为 $\chi(c_i)$ 的代价。

我们选择代价最大的 χ $\chi(c_i)$ 之一。我们计算 χ 中的每个点，并在 H 的对应位置放置一个中心。我们将 $count$ 增加 q'，并相应地更新 $\text{close}(t)$ 和 $\text{cover}(t)$。如果 $t = t_r$，我们再在 v_r 放置一个中心，并将 $count$ 增加 1。

3.3 时间复杂度

定理 1. 仙人掌图上的加权 k-中心问题的可行性测试需要 $O(n)$ 时间，其中 n 是仙人掌图中的顶点数。

证明. 我们只遍历 T 中的每个节点一次。如果节点 t 表示铰点或嫁接点，则处理该节点所需的时间与 t 的子节点数呈线性关系。否则，t 表示一个环，我们将问题转换为圆弧上的问题。这一步的时间同样与该环中的顶点数呈线性关系。最后，求出 χ,$\text{close}(t)$ 和 $\text{cover}(t)$ 所需的时间也与该环中的顶点数呈线性关系。因此，整个算法的运行时间为 $O(n)$。

□ □

4 路径上的参数化剪枝

设 L 是 $\mathbb{R}^2$ 中的 m 条直线的集合，并设 I 表示 L 中这些直线的 $\binom{n}{2}$ 个两两交点的集合。考虑一种类似于第 3 节中定义的可行性测试，该测试以值 γ 作为输入；当 $\gamma \geq \gamma^*$ 时，返回 可行，其中 γ^* 是预定义的阈值。否则返回 不可行。

设 p_1 是 I 中 y 坐标值不可行的最高点，设 p_2 是 I 中 y 坐标值可行的最低点。我们有 Chen 和 Wang [6] 得出的以下结果。

引理 6 (Chen 和 Wang [6])。给定 L，可以在$O((m + \tau) \log m)$时间内找到 p_1 和 p_2，其中 τ是可行性测试的运行时间。

令 P 为一条大小为 m 的路径，其顶点带有权重、边带有长度，如第1节所示。对于 u，$v \in V(P)$ 表示在 $A(P)$ 中用单个中心覆盖 u 和 v 的代价，而 l-成本(u, v) 表示将中心放置在 v 处以覆盖 u 的代价。令 $R_0(P) = \{$l-成本$(u, v) \mid \forall (u, v) \in V(P) \times V(P)\}$，$R_1(P) = \{$代价$(u, v) \mid \forall (u, v) \in V(P) \times V(P)\}$ 和 $R(P) = R_0(P) \cup R_1(P)$。

令 $v_0 \in V(P)$ 为 P 的最左顶点。对于每个顶点 $v_i \in V(P)$，我们在一个 xy 坐标系的 x 轴上，x 坐标为 $d(v_0, v_i)$ 的位置放置一个点 q_i。对于每个顶点 $v_i \in V(P)$，我们在点 q_i 处放置三条直线 l_i^+，l_i^- 和 l_i^∞，其斜率分别为 $+w(v_i)$ $-w(v_i)$ 和 ∞（垂直线）。令 $L_P = \{l_i^+, l_i^-, l_i^\infty \mid \forall v_i \in V(P)\}$，并令 I_P 表示 L_P 中直线的两两交点集合。

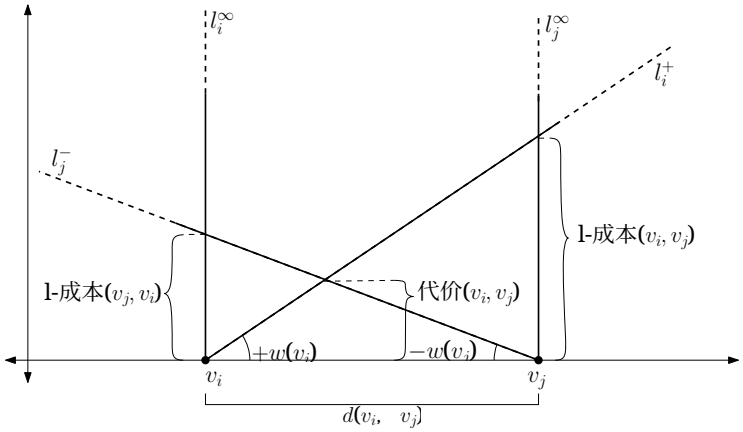


图5：在 $\mathbb{R}^2$ 中表示 $R_0(P) \cup R_2(P)$ 的值。

注意，$R(P)$ 中的所有值都表示为 I_P 中各点的 y 值。参见图5。因此，对于阈值预先设定为 γ^* 的可行性测试，我们可以使用引理 6 找到 λ_1，$\lambda_2 \in \mathbb{R}_{\geq 0}$，使得 $\lambda_1 < \gamma^* \leq \lambda_2$，且 $R(P)$ 中没有值落在范围 (λ_1, λ_2) 内。若有多条路径而非单条路径，也可以得到相同结果。给定多条路径 P_1，P_2，...，P_l，我们依次连接这些路径的端点，将它们拼接成一条路径 P。然后，我们可以对这条合并后的路径执行相同操作，得到 λ_1，$\lambda_2 \in \mathbb{R}_{\geq 0}$，使得 $\lambda_1 < \gamma^* \leq \lambda_2$，且 $\forall \lambda \in \bigcup_{i=1}^{l} R(P_i)$ $\lambda \notin (\lambda_1, \lambda_2)$。因此，我们提出以下论断。

引理 7。给定路径网络 P_1、P_2、.....、P_l以及阈值为 γ^* 的可行性测试，我们可以在 $O((m + \tau) \log m)$ 时间内找到 λ_1 和 λ_2，其中 P 的大小为 m，而 τ 是可行性测试的运行时间。

5 k-中心上的仙人掌茎

本节定义仙人掌茎，它是 Frederickson [7] 为树定义的（路径）茎的推广，并介绍两种用于计算其加权 k-中心的算法。设 S' 是一个仙人掌图，其树表示为路径 P'。假设将 P' 的节点从左到右排序。

S' 的结构是由路径和环交替组成的序列。令P_1、C_1、P_2、C_2.....、P_m、C_m 表示该序列相对于 P'从左到右排序的从左到右排序，其中 P_i 和 C_j 分别表示 S'的一条路径和一个环。注意，路径 P_1 和环 C_m 可能不存在。

如果一个顶点的度为 1，我们就称其为悬挂顶点 。令 S 表示通过向 V中的顶点附加任意数量的悬挂顶点而由 S' 生成的图（S'）。我们称 S 为仙人掌茎。在本节中，我们假设 $|V（S）| = n$。

令 s 和 t 分别为 P' 最左侧和最右侧的节点，并令 g_0 和 h_0为它们在 S' 中各自对应的顶点。我们分别称 g_0 和 h_0 为 S的最左侧 和最右侧 顶点。令 g_i 和 h_i 分别表示 S中环 C_i 的左铰点和右铰点。如果 C_1 的左铰点不存在，则设 g_1为 g_0。类似地，如果环 C_m 的右铰点不存在，则设 h_m 为 h_0。

C_i 的小半球 是 C_i中连接 g_i 和 h_i 的最短路径。 C_i 的大半球 是 C_i中连接 g_i 和 h_i 的另一条路径。如果两条路径长度相同，则任意选取其中一条作为大半球，另一条作为小半球。见图 6。

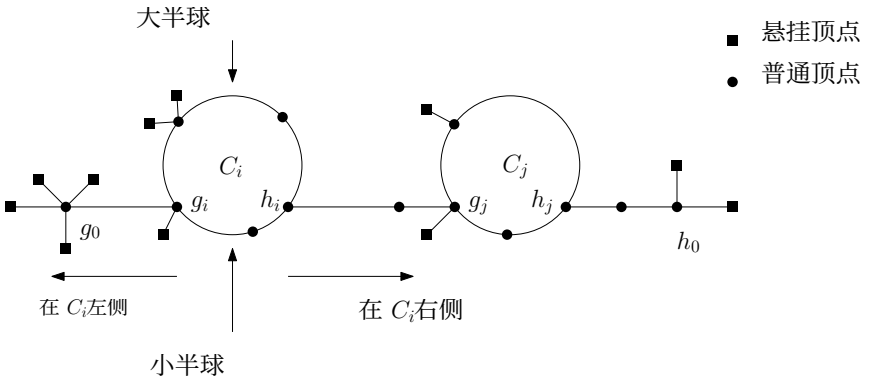


图 6：仙人掌茎。

如果 u 和g_0 之间不存在经过 C_i的路径，则称顶点 $u \in V（S）$ 位于 C_i 的左侧 。类似地，如果 v 和 h_0 之间不存在经过 C_i的路径，则称顶点 $v \in V（S）$ 位于 C_i 的右侧 ^。令 C_i 为将所有与 $V（C_i）$ 相邻的悬挂顶点连接到 C_i 后形成的图。注意，^u， $v \notin V（C_i）$ 。

回顾一下， $A(S)$ 表示 S 所有边上全部点的集合。对于x， $y \in A(S)$，令 π_{xy} 表示 S 中连接 x 和 y 的最短路径。对于 u， $v \in V(S)$，令 $x \in A(S)$ 满足： $w(u) \cdot d(x, u) = w(v) \cdot d(x, v)$，且对于任意两条

在 S, $\pi_{xu} \cap \pi_{xv} = \varnothing$ 中的最短路径 π_{xu} 和 π_{xv}。对于这样的三元组 $(u,\ v,\ x)$，我们将 cost$(u,\ v,\ x)$ 定义为：将 u 和 v 覆盖所需的代价，其中中心位于 x，即 cost$(u,\ v,\ x) = w(u) \cdot d(x,\ u) = w(v) \cdot d(x,\ v)$。我们将 R_S 定义为 cost$(u,\ v,\ x)$ 的所有可能取值组成的集合。

设 $X^* \subset A(S)$ 为关于权重 k-中心问题的仙人掌茎的 k 最优中心集合，并设 λ^* 为其最优代价。我们有以下引理。

引理 8。存在一个三元组（u、v、x），使得 $\lambda^* =$ 代价（u、v、x）。

证明。由于 λ^* 是最优代价，因此存在一个确定 λ^* 的中心 $x \in X^*$。令 $u \in V(S)$ 满足：$w(u) \cdot d(x,\ u) = \lambda^*$，且在 X^* 中距离 u 最近的中心是 x。由于 x 确定了 λ^*，因此 u 始终存在。但仅凭这一点还不足以确定 λ^*，因为我们可以将 x 朝 u 移动，从而降低 x 到 u 的加权距离。因此，存在另一个顶点 $v \in V(S)$，使得 $w(v) \cdot d(x,\ v) = \lambda^*$。此外，$\pi_{ux}$ 与 π_{vx} 不存在重叠路径，因为同样可以将 x 朝某个方向移动，从而降低 x 到 u 和 v 的加权距离。□ □

我们将集合 R_S 称为 λ^* 的候选值集合。根据 u、v 和 x，R_S 中的候选值可分为以下几类。

类型 1： x 不位于 S 的任何环内，

类型 2： x 位于 S 中某个环 C_i 的小半球上，且

（a）： u 和 v 均属于 $V(\hat{C}_i)$，**（b）：** u 或 v 属于 $V(C_i)$，但两者不能同时属于，$\hat{}$
（c）： u 和 v 均不属于 $V(C_i)$。

类型 3： x 位于 S 中某个环 C_j 的大半球上，且

（a）： u 和 v 均属于 $V(\hat{C}_j)$，**（b）：** u 或 v 属于 $V(\hat{C}_j)$，但两者不能同时属于，
（c）： u 和 v 均不属于 $V(\hat{C}_j)$。

除了 R_S 之外，我们还将另一组代价 R'_S 定义为集合 {1-成本$(u,\ v) \mid \forall (u,\ v) \in V(S) \times V(S)$}。在接下来的章节中，我们将介绍用于计算 S 的加权 k-中心的算法。该算法分为两个阶段。在阶段 1 中，我们考察 R_S 中除上述定义的类型 3(c) 之外的所有类型的候选值。在阶段 2 中，我们检查 R_S 中所有类型为 3(c) 的候选值。

5.1 阶段 1

对于 S 的环 C_i，令 A_i 表示其小半球，B_i 表示其大半球。对于给定的仙人掌茎 S，令 S' 表示移除 S 的所有悬挂顶点后剩余的图。参见图 7。

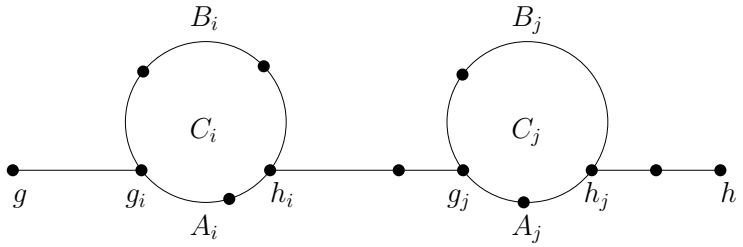


图 7：图 S'，假设 S 为图 6 所表示的图。

通过从 S' 中的每个环 C_i 移除 B_i，并按图 8 所示将一条路径连接到 g_i 和 h_i，我们将 S' 转换为一棵树 $T_{S'}$。请注意，图 8 仅展示了 $T_{S'}$ 的整体结构，部分顶点未予显示。树 T_S 是通过将 S 的所有悬挂顶点连接到它们在 $T_{S'}$ 中各自对应的位置而生成的。参见图 9。

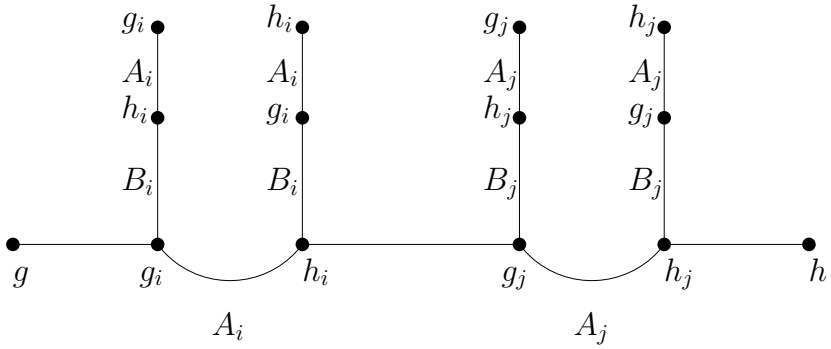


图 8：$T_{S'}$的结构。

基于 T_S 中的顶点，我们在 x-轴上按照如下方式生成 xy坐标系中的点。任意固定关于 g_0 和 h_0 的两个点 p_0 和 q_0，使得 p_0 位于 q_0 的左侧，且两者之间的距离为 $d_{T_S}(g_0,\ g_0)$。对于每个顶点 $u_i \in V(T_{S'})$，我们在 p_0 的右侧放置一个点 p_i，使得两者之间的距离为 $d_{T_S}(g_0,\ u_i)$；并在 q_0 的左侧放置一个点 q_i，使得两者之间的距离为 $d_{T_S}(u_i,\ h_0)$。对于每个顶点 $v_j \in V(T_S) \setminus V(T_{S'})$，其连接到顶点 $u_{j'} \in V(T_{S'})$，我们放置四个点：在$p_{j'}$和 $q_{j'}$的左侧各放置一个点，并在它们的右侧各放置一个点，距离为 $d_{T_S}(u_j,\ v_{j'})$。参见图 10。我们将由所放置点的从左到右排序所表示的路径记为 P_S。

由于，S 中的单个顶点由P_S 中的常数个节点表示，因此我们有 $|V(P_S)| = O(n)$。回想一下，$R_0(P_S) = \{$l-成本$(u,v) \mid \forall (u,v) \in V(P_S) \times V(P_S)\}$ 以及 $R_1(P_S) = \{$代价$(u,v) \mid \forall (u,v) \in V(P_S) \times V(P_S)\}$。注意，$R_0(P_S) = R'_S$ 和 $R_1(P_S)$ 包含了 R_S的所有类型的候选值，但不包括类型 3(c)。我们可以使用引理 7 和第 3 节中的可行性测试来

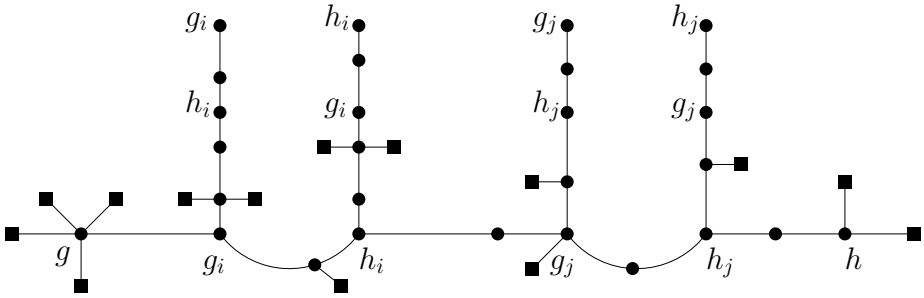


图 9：树 T_S。

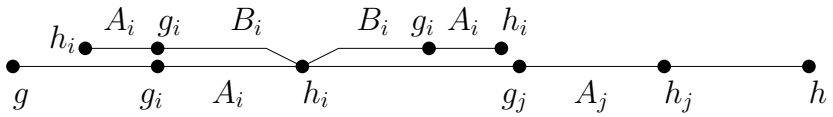


图 10： S' 中的部分顶点映射到实数轴 $\mathbb{R}$上。

求出 λ_1 和 λ_2，使得 $\lambda_1 < \lambda^* \leq \lambda_2$，并且 $R_0(P_S) \cup R_1(P_S)$ 中没有任何值落在范围 (λ_1, λ_2) 内。由于在我们的情形中，m 和 τ 都等于 $O(n)$，这一步需要 $O(n\log n)$ 时间。因此我们可以作出以下断言。

定理 2. *$\lambda^* = \lambda_2$ 是 S 中 k-中心问题的最优代价，前提是 λ^* 不是 R_S的类型 3(c) 候选值。该代价可在 $O(n\log n)$时间内计算，其中 n 是 S的大小。*

定义 1. 顶点 $v \in V$（S）的 λ-覆盖 记为 Φ_λ（v），定义如下：

$$\Phi_\lambda(v) = \{y \in A(S) \mid w(v) \times d(y, v) \leq \lambda\}.$$

如果 $\lambda = \lambda^*$，通常会省略 Φ_λ（$\cdot$）中的下标 λ。

区域 Φ_λ（v）也可定义为 S 中所有可能的位置，在这些位置放置一个中心后，其到 v 的加权距离至多为 λ。

定义 2. 如果 $w(v) \times d(y, v) = \lambda$，则点 $y \in A(S)$ 是 $\Phi_\lambda(v)$ 的端点。通常在 $\lambda = \lambda^*$ 的情况下会省略下标 λ。

$\Phi_\lambda(v)$ 的端点是 S 中所有可能放置中心的位置，使得其到 v 的加权距离等于 λ。参见图11。我们有以下结果。

引理9. 对于 $u \in V(S)$ 和 $e \in E(S)$，当且仅当 $\Phi_\lambda(u)$ 在 e 上恰有一个端点时，$\Phi(u)$ 在 e 上恰有一个端点，对任意 $\lambda \in (\lambda_1, \lambda_2)$ 均成立。

证明。 设 $\Phi(u)$的一个端点位于边 $e = (u_e, v_e) \in E(S)$ $u_e, v_e \in V(S)$上。注意到 l-成本(u, u_e) 和 l-成本(u, v_e)的值都属于 R'_S，因此不在范围（λ_1, λ_2）内。不失一般性，设 l-成本$(u, u_e) \leq$l-成本(u, v_e)。

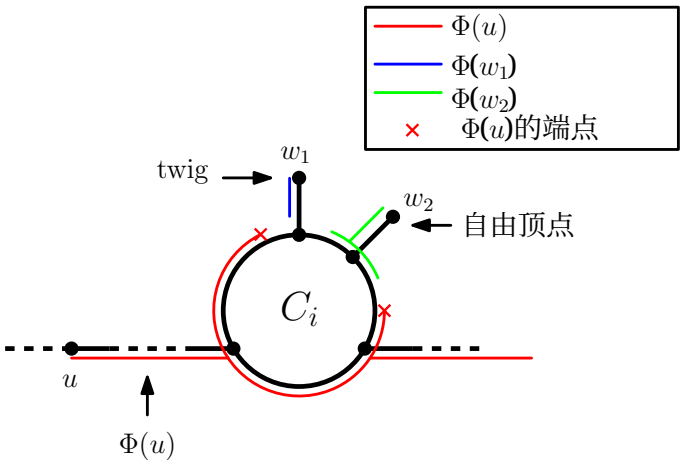


图11：各种定义的可视化。

由于 $\Phi(u)$在 e上恰好有一个端点，因此有 l-成本$(u,\ u_e) < \lambda^* <$l-成本$(u,\ v_e)$。这意味着，对于任意 $\lambda \in (\lambda_1,\ \lambda_2)$，l-成本$(u,\ u_e) < \lambda <$l-成本$(u,\ v_e)$，因此 $\Phi_\lambda(u)$在 e上也恰好有一个端点。另一个方向可以类似证明。□ □

引理10。对于任意两个顶点 $u, v \in V(S)$，设 $e \in E(S)$ 是一条边，且它不属于 S 中位于 u 和 v 之间的任何环的大半球。若且唯若对于所有 $\lambda \in (\lambda_1, \lambda_2)$，区域 $\Phi(u)$ 与 $\Phi(v)$ 在 e 上重叠，当且仅当 $\Phi_\lambda(u)$ 与 $\Phi_\lambda(v)$在 e 上重叠。

证明。令 $e = (u_e,\ v_e)$，其中 $u_e,\ v_e \in V(S)$。令 α^* 表示 $\Phi(u)$ 与 $\Phi(v)$ 在 e 上的重叠区域。同样地，令 α 表示 $\Phi_\lambda(u)$ 与 $\Phi_\lambda(v)$ 在 e 上的重叠区域。我们需要证明，当且仅当 α非空时，α^* 才非空。

如果 α 为空，则 $\Phi(u)$ 或 $\Phi(v)$ 不与 e相交。不妨设是 $\Phi(u)$。那么，l-成本$(u,\ u_e)$，l-成本$(u,\ v_e) \leq \lambda^*$。由于 l-成本$(u,\ u_e)$、l-成本$(u,\ v_e) \notin (\lambda_1,\ \lambda_2)$，这意味着对于任意 $\lambda \in (\lambda_1,\ \lambda_2)$，l-成本$(u,\ u_e)$、l-成本$(u,\ v_e) \leq \lambda$。因此 α 为空。

如果 α^* 非空且包含 u_e，则 l-成本$(u,\ u_e)$，l-成本$(v,\ u_e) \geq \lambda^*$。由于 l-成本$(u,\ u_e)$、l-成本$(v,\ u_e) \notin (\lambda_1,\ \lambda_2)$，这意味着对于任意 $\lambda \in (\lambda_1,\ \lambda_2)$，l-成本$(u,\ u_e)$、l-成本$(v,\ u_e) \geq \lambda$。因此，$\alpha$ 也非空且包含 u_e。关于顶点 v_e，同样可以证明。

如果 α^* 非空且不包含 u_e 或 v_e，那么存在一点 $y_e \in \alpha^*$，使得 $w(u) \cdot d(u, y_e) = w(v) \cdot d(y_e, v)$，并且对于 S中的任意两条最短路径 $\pi_{y_e u}$ 和 $\pi_{y_e v}$，$\pi_{y_e u} \cap \pi_{y_e v} = \varnothing$。这意味着 cost$(u, v, y_e) \in R(S)$。由于 e 不属于 u和 v之间 S 的任何环的大半球，因此 cost(u, v, y_e) 不是前文定义的类型 3(c) 候选值，也不在范围 (λ_1, λ_2) 内。由于 $y_e \in \alpha^*$，cost$(u, v, y_e) \geq$

λ^*，这又意味着对于任意 $\lambda \in (\lambda_1, \lambda_2)$，cost$(u, v, y_e) \geq \lambda$。因此，$\alpha$也非空且包含 y_e。 □ □

5.2 阶段 2

现在我们说明如何确定 λ^* 是否是 R_S的类型 3(c) 候选值。首先，我们给出一个直接的算法，其运行时间为 $O(n^2 \log n)$。随后，在第 5.3 节和第 5.4 节中，我们将介绍两种技术，进一步改善该算法的运行时间。

对于 S 的每个环 C_i，我们按如下方式构造路径 P_{C_i}。回顾一下，g_i 和 h_i 分别是 C_i的左铰点和右铰点。对于 C_j左侧的每个顶点 u_j，我们在一个 xy 坐标系的 x轴上放置一个点 p_j，其 x坐标为 $-d(u_j, g_i)$。类似地，对于 C_i 右侧的每个顶点 $v_{j'}$，我们在 x轴上放置一个点 $q_{j'}$，其 x坐标为 $\Delta_i + d(h_i, v_{j'})$，其中 Δ_i是 C_i的大半球长度。这些点从左到右的排序表示路径 P_{C_i}。

注意，集合 $R' = \bigcup_i R(P_{C_i})$ 包含 R_S 的所有类型 3(c) 候选值。因此，我们可以使用引理 7 找到 λ'_1 和 λ'_2，使得 $\lambda'_1 < \lambda^* \leq \lambda'_2$，并且 R' 中没有任何值位于范围（λ'_1, λ'_2）内。我们将之前的 λ_1 和 λ_2 值更新为 $\lambda_1 \leftarrow \max\{\lambda_1, \lambda'_1\}$ 和 $\lambda_2 \leftarrow \min\{\lambda_2, \lambda'_2\}$。由于现在 R_S 中没有任何值位于范围（λ_1, λ_2）内，因此有 $\lambda^* = \lambda_2$。

ˆ令 P 表示对于 S 中的所有环 C_i，将路径 P_{C_i} 进行拼接所得的结果。由于每条路径 P_{C_i} 的大小都可以是 $O(n)$，因此 ˆP 的大小可以是 $O(n^2)$，而我们的算法的运行时间为 $O(n^2 \log n)$。

5.3 对阶段 2 的改进

改进阶段 2 运行时间的一种方法是，不对所有可能的类型 3(c) 候选值进行搜索，而只搜索绝对必要的候选值。我们通过对 S 执行可行性测试并最优地放置中心来实现这一点。每当我们无法确定新中心的位置时，就额外执行一次可行性测试。

我们首先选择一个 $\lambda \in$ （λ_1, λ_2）（来自阶段 1），然后对 S 执行可行性测试。该测试从左到右处理 S 的顶点。在任一中间步骤中，可行性测试都可能在 S 上放置中心。如果这些中心被放置在不属于 S 中任何环的大半球的边上，我们就继续放置这些中心。这是因为，如果任何此类中心 x_i 确定了最优代价 λ^*，那么该代价就不属于范围 (λ_1, λ_2)，因为此时 λ^* 不是类型 3(c) 候选值。因此，在最优情况下，由 x_i 覆盖的顶点将保持不变。

如果中心被放置在某个环 C_j of S 的大半球上的边上，那么我们还需执行一些额外步骤。设 x_l 是这样的一个中心。x_l 有可能决定最优代价 λ^*。设 u^*, $v^* \in V(S)$ 是 x_l 的两个临界顶点，即 cost$(u^*, v^*, x_l) = \lambda^*$。如果 u^* 和 v^* 均不属于 $V(C_j)$，则值 cost(u^*, v^*, x_l) ˆ 是一个类型 3(c) 候选值。这意味着 cost(u^*, v^*, x_l) 可能位于范围 (λ_1, λ_2) 内，因此我们无法提供

与前一种情况相同的保证。因此，我们尝试进一步缩小范围 (λ_1, λ_2)，如下所示。

我们生成前面提到的路径 P_{C_j}，并使用引理 7 找到 λ_1' 和 λ_2'，使得 $\lambda_1' < \lambda^* \leq \lambda_2'$，且 $R(P_{C_j})$ 中没有任何值落在范围 (λ_1', λ_2') 内。我们将之前的 λ_1 和 λ_2 值更新为 $\lambda_1 \leftarrow$ 最大值$\{\lambda_1, \lambda_1'\}$ 和$\lambda_2 \leftarrow$ 最小值$\{\lambda_2, \lambda_2'\}$。

这确保对于任意 $u, v \in V(S)$，代价$(u, v, x_l) \notin (\lambda_1, \lambda_2)$。我们将对 λ 的选择更新为新区间 (λ_1, λ_2)，并继续使用我们的可行性测试算法放置中心。在此步骤之后放置的中心 x_l 将确保由其覆盖的顶点在最优情况下保持不变。我们继续这一过程，直到所有 k 个中心都已放置。我们提出以下断言。

引理 11. 阶段 *2* 完成后，$\lambda^* = \lambda_2$。

证明。我们需要考虑的 $R(S)$ 的类型 3(c) 候选值只有 cost(u, v, x)，其中 x 位于一个环的大半球上，而我们的可行性测试希望在该处放置一个中心。由于阶段 2 完成后，没有这样的值落在范围 (λ_1, λ_2) 内，因此 $\lambda_2 = \lambda^*$。□ □

引理 12. 阶段 *2* 所需的时间至多为 $O(kn\log n)$。

证明。每当需要在某个环 C_j 的大半球上为 S 放置一个中心时，我们就生成路径 P_{C_j}，并计算适当的 λ_1' 和 λ_2'。与阶段 1 类似，此步骤耗时 $O(n \log n)$，因为 $|V(P_{C_j})| = O(n)$。由于我们在 S 中放置 k 个中心，因此此步骤至多重复 k 次。因此，运行时间为 $O(kn\log n)$。 □ □

5.4 进一步改进阶段 2

5.4.1 预备知识

考虑对 S 进行可行性测试，其中 $\lambda = \lambda^*$ 为 S 的最右顶点，h_0 为根。令 $X^* \subset A(S)$ 表示由该可行性测试放置的 k 中心集合。集合 X^* 是 S 中加权 k-中心问题的最优解。

由于我们假设 λ^* 是 R_S 的类型 3(c) 候选值，因此存在 $u^*, v^* \in V(S)$ 和 $x^* \in X^*$，使得 $\lambda^* =$ 代价(u^*, v^*, x^*)。此外，x^* 位于某个环 C_{i^*} 的大半球上，并且 $u^*, v^* \notin V(C_{i^*})$。我们考虑三元组 (u^*, v^*, x^*)，其中 C_{i^*} 是 S 中最左侧的可能环。

我们假设 $\Phi(u^*)$和 $\Phi(v^*)$完全覆盖环 C_{i^*} 的小半球，如图12所示，并展示一些性质。

通过类似于引理9的证明的论证，我们可以证明：对于某个$v \in V(S)$，$\Phi(v)$完全覆盖 S 的环 C_i 的小半球，当且仅当 $\Phi_{\lambda_2}(v)$也完全覆盖该小半球。

定义 3. 枝 是任意一个悬挂（度为一）顶点 u，它与顶点v相邻，且满足 $v \notin \Phi(u)$，$u, v \in V(S)$。

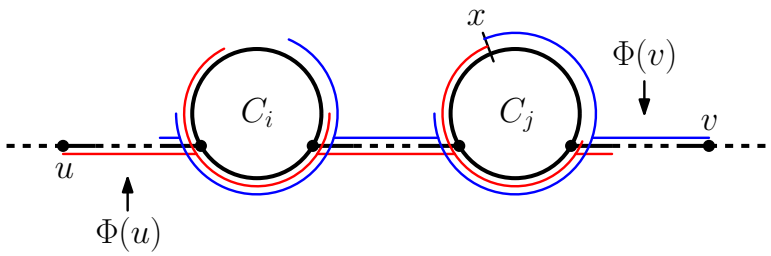


图 12： λ的区域，其中 u和 v满足 $\lambda =$代价(u， v， x)。

枝是 S的任意悬挂顶点 u，对于该顶点，若将中心 x 放置在其相邻顶点处，则它到 u的加权距离大于 λ^*。按照引理9的证明中的类似论证，我们还可以证明，若 $v \notin \Phi_{\lambda_2}(u)$，则 u 是枝。

定义4。 若对于$V(S)$中的所有枝条 v， u 满足 $\Phi(u) \cap \Phi(v) = \varnothing$，则称顶点 u 为自由顶点。

自由顶点是 S 中无法与某个枝条同时由单个中心覆盖的任意顶点。参见图11。

定义5。 环 C_i的 λ-相交，记作 $\Theta_\lambda(C_i)$，定义为

$$\Theta_\lambda(C_i) = \bigcap_{v \in V_i'} \Phi_\lambda(v),$$

其中， V_i'表示 $V(\hat{C}_i)$中所有自由顶点构成的集合。同样，若 $\lambda = \lambda^*$，则省略 $\Theta_\lambda(\cdot)$中的下标 λ。

参见图13。现在我们可以提出以下论断。

引理13。 对于顶点 u^* 和环 C_{i^*}， 以下两个条件成立： $(\mathrm{a})\Phi(u^*) \cap \Theta(C_{i^*}) \neq \varnothing$ 以及(b) $h_{i^*} \notin \Theta(C_{i^*})$。

证明。 首先，我们证明 V'_{i^*} 中的所有顶点都由中心 x^*覆盖。假设还存在另一个中心 x'， 覆盖了 V'_{i^*} 中一些未被 x^*覆盖的顶点。由于 V'_{i^*} 中的所有顶点都是非枝叶顶点，因此 x' 可以放置在 C_{i^*} 上（而不是其任何关联边内部）。但是， $\Phi(u) \cup \Phi(v)$ 与 C_{i^*} 完全重叠，因此 x' 也会覆盖 u^* 或 v^*。这意味着 cost(u^*， v^*， x^*) 将无法确定 λ^*，这与假设矛盾。

由于 V'_{i^*} 中的所有顶点以及顶点 u^* 都由 x^* 覆盖，这意味着 $\Theta(C_{i^*})$ 非空，并且与 $\Phi(u^*)$ 相交，从而证明了我们的第一个条件。如果 $\Theta(C_{i^*})$ 包含 h_{i^*} ，那么可行性测试会将 x^* 放置在 h_{i^*} 上，而不是放置在 C_{i^*} 的大半球上。这将导致 cost(u^*, v^*, x^*) 不属于类型 3(c) 候选值，从而产生另一个矛盾。这证明了我们的第二个条件。 □ □

引理14. 没有环 C_j 位于 u^* 和 C_{i^*} 之间，并且同时满足引理*13*的两个条件。

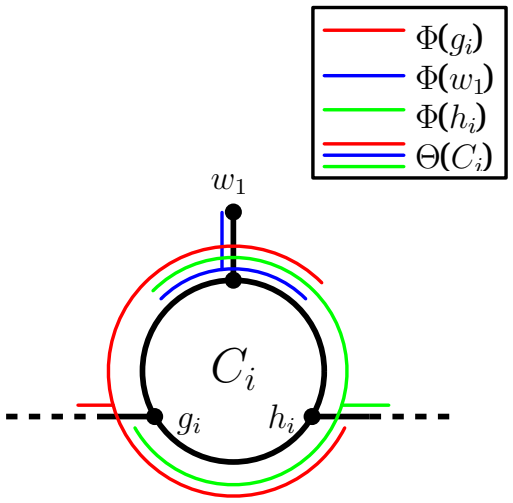


图13： λ^*与 C_i的相交。

证明. 假设存在一个环 C_j，位于顶点 u^* 和环C_{i^*} 之间，并且同时满足引理13的两个条件。由于 u^* 由位于 C_j右侧的 x^* 覆盖， V'_j 中的所有顶点要么由 x^* 覆盖，要么由放置在 C_j上的至多一个中心覆盖。该条件成立，因为 $\Theta(C_j)$ 非空，且 V_j 没有枝条。

如果 V'_j 中的所有顶点都被 x^* 覆盖，那么 $h_j \in \Theta(C_j)$，这就产生了矛盾。如果 $h_j \notin \Theta(C_j)$，那么需要将一个中心（例如 x' ）放置在 C_j 上，以覆盖 V'_j 中的顶点。但是，由于 $\Phi(u^*)$与 $\Theta(C_j)$相交，在处理 C_j时，可行性测试会以一种也覆盖 u^*的方式放置 x' 。这与此矛盾，因为此时 cost(u^*, v^*, x^*) 将不会确定 λ^*。□

根据引理13和14，我们可以作出如下观察。

观察 1。 设 u、 $v \in V(S)$ 和 x 属于环 C_i 的大半球，且 $\Phi(u)$ 与 C_i 的小半球完全重叠。我们只考虑类型 3(c) 候选值 cost$(u$、 v、 $x)$，其中 C_i 是 u 右侧满足引理 13 中两个条件的第一个环。

考虑一个 xy 坐标系 $\mathbb{R}^2$。对于每个顶点 $v_i \in V(S)$，我们在 $\mathbb{R}^2$ 的 x 轴上、x 坐标为 $d(v_i$、 $h_0)$ 的位置放置一个点 p_i 。对于每个这样的点p_i ，我们在其上放置一条斜率为 $w(v_i)$ 的直线 l_i 。令 L_S 为这些直线的集合。我们给出如下定义。

定义6。 顶点 $v_i \in V(S)$ 的 秩 是直线 l_i 与 L_S 中所有直线的交点在水平线 $y = \lambda^*$ 上从左到右排序后所处的位置。

注意，用 λ_2 替换 λ^* 后，顶点的秩不会改变。

设 $u, v \in V(S)$ 满足 $\text{rank}(u) \leq \text{rank}(v)$。设 C_i 是位于 u 和 v 右侧的一个环。根据秩的定义，我们可以得到以下观察。

观察2。 $\Phi(u) \cap \mathcal{A}(C_i) \subseteq \Phi(v) \cap \mathcal{A}(C_i)$。

5.4.2 寻找第一个环。

我们给出一种算法，用于为每个顶点 $v \in V(C)$ 找到其右侧第一个满足引理13的环（如果存在）。设 $\mathcal{C}$ 为 S 中按从左到右顺序排列的环集合。我们计算 $\Theta_{\lambda_2}(C_i)$，其中所有环 C_i 都属于 $\mathcal{C}$。我们从 $\mathcal{C}$ 中移除满足以下任一条件的环 C_j： $\Theta_{\lambda_2}(C_j) = \varnothing$ 或 $h_j \in \Theta_{\lambda_2}(C_j)$。这是因为对于任意点 $x \in A(C_j)$，类型 3(c) 候选值 $\text{cost}(u, v, x)$ 永远无法满足引理13。

我们按秩递增的顺序处理 $V(S)$ 中的顶点，并执行以下步骤。设 v_i 为当前正在处理的顶点。找到位于 u_i 右侧最靠左的环 $C_j \in \mathcal{C}$。如果不存在这样的环，则跳过 u_i，继续处理秩更高的顶点。这意味着 u_i 右侧不存在满足引理13的环。

如果存在环 C_j，那么我们检查 $\Phi_{\lambda_2}(u_i) \cap \Theta_{\lambda_2}(C_j) \neq \varnothing$ 是否成立。如果该条件成立，我们就能正确识别出满足引理13的、位于 u_i 右侧的第一个顶点 C_j。接着处理秩更高的下一个顶点。

否则，如果该条件不成立，我们就从 $\mathcal{C}$ 中移除 C_j。这是因为根据观察2，对于任意满足 $\text{rank}(v_j) \geq \text{rank}(v_i)$ 的顶点 v_j，$\Phi_{\lambda_2}(u_i) \cap \Theta_{\lambda_2}(C_j) = \varnothing$。我们继续处理 C_j 右侧的 $\mathcal{C}$ 中的下一个环。当 $\mathcal{C}$ 为空，或 $V(S)$ 中的所有顶点都已处理完毕时，我们停止。

当一个顶点成功映射到某个环时，我们将该操作的开销计入该顶点；当某个环与一个顶点进行映射比较但未成功时，我们将该操作的开销计入该环。由于每个顶点和每个环恰好计入一次，该过程的运行时间为 $O(n \log n)$。其中的 $\log n$ 因子来自于我们必须找到 $\mathcal{C}$ 中位于某个特定顶点右侧的第一个环，这需要 $O(\log n)$ 时间。

5.4.3 改进的算法。

下面给出最终算法，该算法显著改善了阶段 2 的运行时间。考虑一个 xy-坐标系 $\mathbb{R}^2$。对于每个顶点 $v_i \in V(S)$，我们在 $\mathbb{R}^2$ 的 x-轴上、原点左侧距离为 $d(v_i, h_0)$ 处放置一个点 p_i。同样地，对于每个顶点 $v_i \in V(S)$，令 C_{j_1} 为 v_i 右侧满足引理13的第一个环，并令 C_{j_2} 为 v_i 右侧的环，其中 $\Phi(v_i)$ 的一个端点位于其小半球上。我们分别在 x-轴上、原点左侧距离为 $d(v_i, g_{j_1}) + \Delta_{j_1} + d(h_{j_1}, h_0)$ 和 $d(v_i, g_{j_2}) + \Delta_{j_2} + d(h_{j_2}, h_0)$ 处放置点 $p_{j'_1}$ 和 $p_{j'_2}$，其中 Δ_i 是环 C_i 的大半球长度。

令 P'_S 为这些点按从左到右排序后形成的路径。我们可以作出如下观察。

观察 3. R_S 的每个相关类型 3(c) 候选值都包含在$R(P'_S)$中。

因此，利用引理 7，我们可以找到 λ'_1 和 λ'_2，使得 (a) $\lambda'_1 < \lambda^* \leq \lambda_2$，且 (b) 在范围 ($\lambda'_1$, λ'_2) 内不存在相关的类型 3(c) 候选值。最后，我们更新 $\lambda_1 \leftarrow \max\{\lambda_1, \lambda'_1\}$和 $\lambda_2 \leftarrow \min\{\lambda_2, \lambda'_2\}$。由于我们已穷尽 λ^*的所有候选值，因此 $\lambda^* = \lambda_2$。由于 P'_S 的大小为 $O(n)$，我们得到以下定理。

定理 3. 在大小为 n 的仙人掌茎上，带权 k-中心问题可以在 $O(n\log n)$ 时间内求解。

6 主算法

现在我们可以介绍仙人掌图 G上 k-中心问题的算法。初始化 $\lambda_1 \leftarrow 0$ 和 $\lambda_2 \leftarrow \infty$。令 G' 为从 G 中移除所有悬挂顶点后得到的图，并令 T 为其树表示。注意， t_r （一个叶节点）及其关联顶点 v_r 分别表示 T 和 G的根。

定义 7。 有根树的一个最大路径 是树中的任意子路径，其端节点的度为 1 或 3 及以上，且内部节点的度为 2。叶最大路径 是包含树的一个叶节点（根除外）的任意最大路径。

对于 T 的每个叶最大路径 π_i，我们执行以下操作。令 t_1、t_2、.....、t_m 为 π_i 中满足 t_1 为叶节点的节点。如果 t_m 表示 G 的一个环，则将 π'_i 设为删除节点 t_m 后的路径 π_i；否则， π'_i 与 π_i 相同。令 S'_i 为 G 所表示的 π'_i 的子图，并令 S_i 为通过将 G 中与 S'_i 相邻的所有悬挂顶点附加到该子图上所形成的图。如果 π'_i 包含节点 t_m，则 t_m 要么是嫁接点，要么是铰点。令 $v_m \in V(G)$ 为 t_m 的关联顶点。我们不将与 v_m 相邻的悬挂顶点包含在 S_i 中。我们称 S_i 为 G 的叶仙人掌茎 。请参阅图 14 和图 15。

对于所有叶仙人掌茎 S_i，我们执行第 5 节中的阶段 1 和阶段 2。在阶段 1 中，我们为每个 S_i生成 P_{S_i}。我们使用引理 7 找到对应的λ'_1 和 λ'_2 值。在阶段 2 中，我们可以依次处理每个 S_i，以找到更新后的 λ'_1 和 λ'_2 值。可以证明，$\lambda'_1 < \lambda^* \leq \lambda'_2$，且$\bigcup_i R_{S_i}$ 中没有任何值落在范围（λ'_1, λ'_2）内。我们将 λ_1 和 λ_2 分别更新为 $\lambda_1 \leftarrow \max\{\lambda_1, \lambda'_1\}$和 $\lambda_2 \leftarrow \min\{\lambda_2, \lambda'_2\}$。

现在，我们通过选择一个 $\lambda \in$ （λ_1, λ_2），对 G 进行可行性测试。令 X 为可行性测试所放置的中心集合。这就在 G 上定义了一个划分，其中每个顶点都与其最近中心相关联。根据我们对 λ_1 和λ_2的定义，对于 G中的每条仙人掌茎 S_i， X 在 S_i 上定义的划分与最优中心集合 X^*所定义的划分相同。Frederickson 等人 [8] 在设计其最优算法时，充分利用了这一性质。我们按如下方式处理 G 的每条叶仙人掌茎 S_i，并将其转换为 G的一个悬挂顶点。

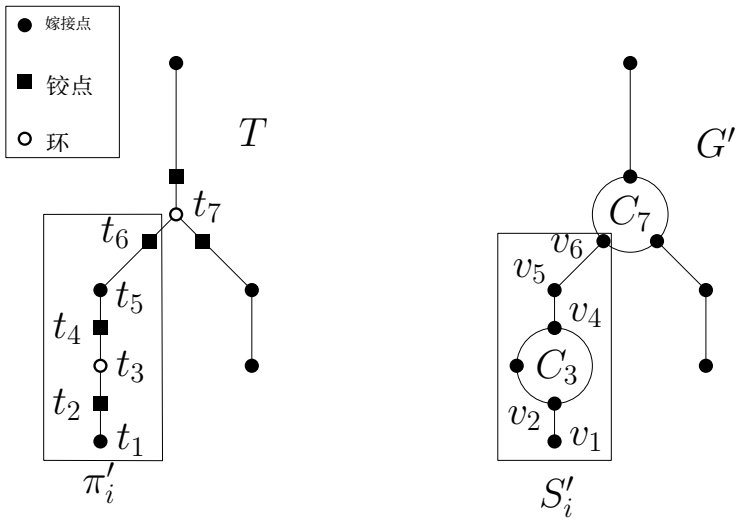


图14：从 t_1 到 t_7 的节点表示 T的叶最大路径 π_i 。子路径 π_i' 从 t_1 到 t_6，其中 t_7 表示 C_7。子图 S_i' 从 v_1 到v_6。

令 $h \in V(S_i)$ 为 S_i 与图 G其余部分相连的顶点。令 $x_h \in X$ 为距离 h 最近的中心，并令 $V' \subseteq V(S)$ 为以 x_h 为最近中心的顶点集合。我们将 S 替换为一个连接到 h 的悬挂顶点 h'，使得 $h' = \arg\min_v\{\lambda/w(v) - d(h, v) \mid \forall v \in V'\}$，并将边 (h, h') 的长度设为 $d(h, h')$。令 $X_{S_i} \subseteq X$ 为在 S_i中放置的中心集合。由于 $\lambda \in (\lambda_1, \lambda_2)$，在最优情况下，$V(S_i)$ 中的顶点 $\setminus V'$ 也会被 $X_{S_i} \setminus \{x_h\}$ 覆盖。因此，我们可以确定这些中心也应包含在最优集合中。

我们将已放置的中心数从 k 中减去，即 $|X_{S_i} \setminus \{x_h\}|$。我们在新的仙人掌图中递归地继续这一过程，直到只剩下一条仙人掌茎。我们可以使用第 5 节中的算法计算这条茎中的加权 k-中心。由此得到以下结果。

定理4。 仙人掌图 G 上的加权 k-中心问题可以在$O(n$对数$^2\, n)$ 时间内求解，其中 n 是 G的顶点数。

证明。 每次迭代中，执行阶段 1 和阶段 2 的步骤需要$O(n\log\, n)$时间。此外，在每次迭代中，图中所有叶仙人掌茎都会缩减为一个悬挂顶点。这可能会使新的仙人掌茎成为叶仙人掌茎。然而，这类新仙人掌茎的数量最多为此前叶仙人掌茎数量的一半。因此，迭代次数至多为$O(\log\, n)$。所以，我们的算法在所有迭代中所需的总时间为 $O(n\log^2 n)$。

□ □

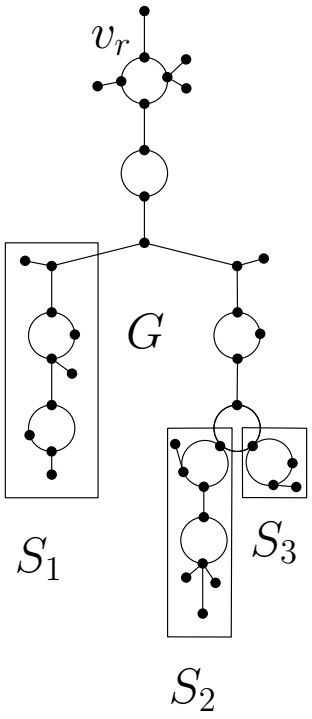


图15： S_1， S_2 G 的 S_3 仙人掌图叶茎，根为 v_r。

7 结论

本文提出了一种 $O(n\log^2 n)$时间算法，用于求解规模为 n的仙人掌图上的加权k-中心问题。我们改造并采用了 Frederickson 和 Johnson 的 [9]参数搜索技术，该技术由 Wang 和 Zhang [13],进行了推广，从而得到我们的结果。他们在论文中还展示了如何通过迭代优化可行性测试算法，实现次线性运行时间。值得进一步研究如何改造这一思路，为我们的问题找到更快的可行性测试算法。我们认为，这有望将算法的运行时间进一步缩短一个 $\log n$ 因子，而这将是最优的。

参考文献

[1] Aritra Banik、Binay Bhattacharya、Sandip Das、Tsunehiko Kameda 和 Zhao Song。重新研究树网络中的 p-中心问题。载于第十五届斯堪的纳维亚算法理论研讨会及专题会议，2016 年。[2] Boaz Ben-Moshe、Binay Bhattacharya、Qiaosheng Shi 和 Arie Tamir。仙人掌图网络中中心问题的高效算法。理论计算机科学，378(3)：237–252，2007 年。

[3]Binay Bhattacharya、Sandip Das 和 Subhadeep Ranjan Dev。固定 k下树中的加权 k-中心问题。载于第三十届国际算法与计算研讨会（*ISAAC 2019*）。Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik，2019 年。[4]Binay Bhattacharya、Mordecai J Golin、Yuya Higashikawa、Tsunehiko Kameda 和 Naoki Katoh。动态流路径网络中计算 k-汇的改进算法。载于算法与数据结构研讨会，第 133–144 页。施普林格，2017 年。[5] Binay Bhattacharya 和 Qiaosheng Shi。网络p-中心定位问题的改进算法。计算几何，47(2)：307–315，2014 年。[6]Danny Z Chen 和 Haitao Wang。关于直线排列搜索及其应用的注记。信息处理快报，113(14-16)：518–521，2013 年。[7] Greg N Frederickson。树划分的最优算法。载于第二届年度 *ACM-SIAM* 离散算法研讨会论文集，第 168–177 页，1991 年。[8] Greg N Frederickson。参数搜索与树中供应中心的定位。载于算法与数据结构研讨会，第 299–319 页。施普林格，1991 年。[9] Greg N Frederickson 和 Donald B Johnson。通过生成和搜索优良数据结构来寻找 k阶路径和 p-中心。算法杂志，4(1)：61–80，1983 年。[10] Alexander Gavruskin、Bakhadyr Khoussainov、Mikhail Kokho 和 Jiamou Liu。单调区间调度问题的动态算法。理论计算机科学，562：227–242，2015 年。[11] Wen-Lian Hsu 和 Kuo-Hui Tsai。圆弧图上的线性时间算法。信息处理快报，40(3)：123–129，1991 年。[12] Oded Kariv 和 S Louis Hakimi。网络定位问题的算法方法。I：p-中心。*SIAM*应用数学杂志，37(3)：513–538，1979 年。[13] Haitao Wang 和 Jingru Zhang。树中k-中心问题的 $O(n\log\ n)$时间算法。载于第三十四届国际计算几何研讨会（*SoCG 2018*）。Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik，2018 年。