arXiv:2601.04626v1 [cs.DS] 8 Jan 2026

利用射线投射查询实现 RDV 图支配集问题的次线性算法*

Therese Biedl[1][0000-0002-9003-3783] 和 Prashant Gokhale[2]

[1] David R. Cheriton 计算机科学学院，滑铁卢大学，加拿大安大略省滑铁卢，biedl@uwaterloo.ca[2] 美国威斯康星州麦迪逊，威斯康星大学麦迪逊分校，prashant.gokhale@wisc.edu

摘要。 本文研究 $RDVgraphs$ 中的支配集问题。 $RDVgraphs$ 是介于区间图与弦图之间的一类图，定义为根树中向下路径的**顶**点交图。在先前的一篇论文中，研究者证明，RDV 图中的邻接查询可以转化为判断一条水平线段是否与一条垂直线段相交的问题。随后，他们利用优先搜索树，在一个含 n 个顶点的 RDV 图中，以 $O(n\log n)$ 时间求得最大匹配，也就是说，无须检查所有边。本文证明，若进一步使用射线发射数据结构，则还可以在 $O(n\log n)$ 时间内求得 RDV 图的最小支配集（假设给定该图的线性规模表示）。同样的思路也可用于给出一种新证明，以 $O(n)$ 时间求得区间图的最小支配集。

1 引言

在图 G 中，顶点的一个子集 S 称为 支配 集，当且仅当图中的每个顶点要么属于 S，要么与 S 中的某个顶点相邻。求最小支配集是图论和图算法中历史最悠久的问题之一，其应用包括无线网络中的路由问题和设施选址问题。一般而言，求最小支配集是 NP 困难问题 [13]，因为可以通过集合覆盖问题直接归约得到。另一方面，已有一种简单易知的 $O(\log n)$-近似算法 [7]。

本文研究如何在结构特征使得能够高效求解最小支配集的图类中，寻找最小支配集这一问题。对于弦图，最小支配集问题仍然是 NP困难 [3]，但对于排列图[12]、余图 [8], 和强弦图 [11], 则存在多项式时间算法。（第 2 节将回顾后文需要用到的定义。）关于完美图各种子类中最小支配集问题复杂度的更多信息，感兴趣的读者可参阅 Corneil 和 Stewart [9]撰写的综述论文。

*TB 的研究得到加拿大自然科学与工程研究委员会资助（RGPIN-2020-03958）。PG 的研究得到 MITACS Globalink 研究生奖学金资助。相关成果也作为第二作者硕士论文的一部分发表 [17]，论文完成于滑铁卢大学。

本文关注的是寻找用于求解最小支配集的亚线性时间算法，也就是说，我们甚至不能查看所有边。这里的关键在于，图是以隐式方式给出的，通常通过交表示来描述。例如，在区间图（即 n一维区间的交图）中，如果给定合适的区间表示，就可以在 $O(n)$ 时间内找到一个最小支配集，尽管该图可能有 $\Theta(n^2)$ 条边。[3] 区间图中的其他问题也可以在亚线性时间内求解，例如，路径覆盖、哈密顿回路和支配划分见 [5] ，最大匹配见 [20]。Cheng、Kaminski 和 Zaks [6]将区间图中支配集问题的算法扩展到了一个略大于区间图的图类，具体而言，是蜘蛛（由多条路径共享同一个节点构成的树）中向下路径的交图。

本文研究一种称为*RDV* 图的图类，它严格包含 Cheng、Kaminski 和 Zaks 的图类（因此也包含区间图）。RDV 图是任意有根树中向下路径的交图。已知这类图是强弦图，因此我们可以在线性时间内求出其最小支配集 [11]。事实上，这一结论早已有之：Booth 和 Johnson 证明，对 RDV 图应用一种直接的线性时间贪心算法即可得到最优解 [3]。

本文旨在改进 Booth 和 Johnson 的贪心算法，使其运行时间达到 $O(|T| + n \log n)$，其中 $|T|$ 是给定 RDV 图交表示中宿主树的规模。（我们可以假设 $|T| \in O(n)$ [2],，因此通常将时间界写作 $O(n \log n)$。）虽然这还不及区间图算法快，但仍然是次线性的，因为 RDV 图可能有二次数量的边。关键在于这样一个观察（首次发表于一篇相关论文 [2]）：RDV 图中的邻接查询可以转化为水平线段与垂直线段之间的交点问题。在 [2], 中，这一观察与范围最小值查询（通过优先搜索树实现）相结合，得到了用于求 RDV 图最大匹配的次线性算法。本文将证明，如果进一步还使用射线射击数据结构，那么无需遍历所有边，也能高效解决更复杂的邻域查询，并以此在次线性时间内求出最小支配集。（这一技术应有更广泛的适用性；我们还将简要说明，也可以用它求出区间图的最小支配集。）

本文结构如下：第 2 节在回顾一些定义之后，简要介绍配套论文中的相关结果。第 3 节给出贪心算法，并解释其哪些操作难以利用已知技术加速。接着，我们说明如何使用射线投射数据结构来实现该操作。由此可为支配集的贪心算法带来更快的运行时间。我们讨论一种在 $O(n)$ 时间内运行的

[3] 这一结果归功于 [1,4,26] ，发表于 [6], ，但这三篇论文均未声称存在一个用于区间图支配集的 $O(n)$ 时间算法。最接近的是 [1],，其中给出的最小权全支配集问题运行时间为 $O(n \log n)$。不过，很容易设计出一个 $O(n)$ 算法；另见第 4 节。

区间图支配集算法，并在第 5 节以进一步的思考作结。

2 背景

我们假设读者熟悉图的相关知识，基本术语可参见 Diestel 的著作 [10]。对于图中的顶点 z，我们用 $N[z]$ 表示其闭邻域，即该顶点 z 及其所有邻居。通常，我们会固定一个顶点顺序$z_1, \ldots, z_n$。此时，我们用 $N_{\geq}[z_i]$ 表示闭邻域中满足条件 $q \geq i$ 的顶点 z_q（类似地定义 $N_{<}[z_i], N_{\leq}[z_i], N_{>}[z_i]$）。

本文仅研究具有作为树的子树的顶点交图的表示（简称树交表示）的图 G。这种表示由一个宿主树 T 以及对图 G的每个顶点 z，一个子树 $T(z)$（属于 T）组成；当且仅当 z, w 是图 G 的一条边时，$T(z)$ 和 $T(w)$ 至少共享 T 的一个节点。按照惯例，我们用“节点”指称 T 的顶点，以便与 G 的顶点区分。众所周知 [14]，一个图具有树交表示，当且仅当它是弦图，即不含长度为 4 或更大的诱导圈。

通过以某种方式限制宿主树和/或子树 $T(z)$，可以定义弦图的许多子类。例如，我们提到过 Cheng、Kaminski 和 Zaks [6] 研究的一个（未命名）图类：其中宿主树被限制为蜘蛛，子树被限制为不包含蜘蛛中心作为内部顶点的路径。本文研究的图类由 Monma 和 Wei [23], 命名，不过 Gavril [15]更早就已对其进行研究：

定义1。有根有向路径图（或RDV图 *[23]*）是具有RDV 表示的图，即具有有根宿主树 T 的树交表示，其中每个顶点 z 对应的子树 $T(z)$ 都是向下路径，也就是说，这种路径从某个节点出发，之后始终沿着 T向下延伸。

我们可以假设 T 有 $O(n)$ 个节点 [2]。全文中，我们假设 RDV图 G 是通过一个 RDV 表示 (T, t, b) 隐式给出的；也就是说，给定一棵有根树 T，以及对于 G 中的每个顶点 z，给定一个顶节点 $t(z)$和一个底部节点 $b(z)$，其中后者是前者的后代节点 $t(z)$；并约定 $T(z)$是连接 $t(z)$与 $b(z)$的路径。对 G 进行自底向上枚举，即按 $t(z)$到根节点的距离递减排列顶点，距离相同时任意排序。该过程可通过桶排序在 $O(|T| + n)$ 时间内完成。

2.1 从 RDV 表示到线段

我们简要回顾一下伴随论文 [2] 中的一些重要见解。令G 为一个具有 RDV 表示的图，其宿主树为 T。按如下方式为宿主树 T 的节点赋点：对于每个节点 ν，定义其 y 坐标 $y(\nu)$为节点 ν 到 T根节点的距离。要定义 x 坐标，先为每个节点的子节点任意确定一个顺序，然后将 T的叶节点

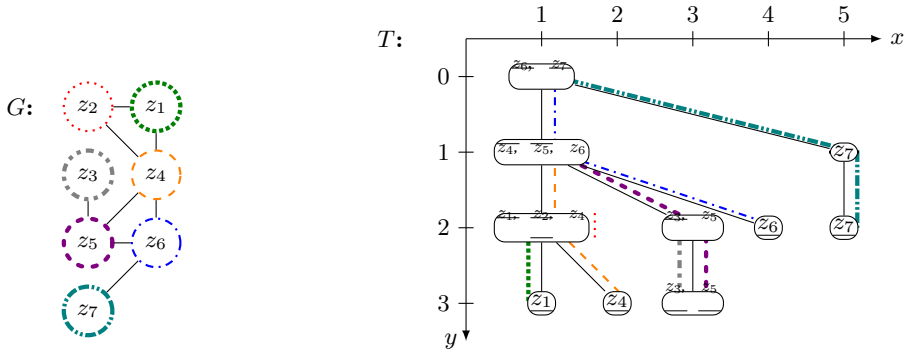

图 1. 一个带有 RDV 表示的 RDV图。每个节点 ν列出满足 $\nu \in T(z)$ 的顶点 z；如果 $\nu = b(z)$，我们记作 z；如果 $\nu = t(z)$，则记作 z。我们还用折线表示路径，颜色/虚线样式与顶点相对应。节点绘制在其坐标处，索引对应自底向上枚举顺序。

从左到右依次编号为 $L_1, \ldots, L_\ell$，然后对每个节点 ν，令 $x(\nu)$ 为 ν的后代节点中最左侧叶节点的序号。图 1 中，每个节点ν 都绘制在点 $(x(\nu)\ y(\nu))$ 处（其中 y坐标从上到下递增）。我们可以在 $O(|T|)$ 时间内计算出这些坐标。

定义 2. 设 G 是一个具有 *RDV* 表示的图。对于每个顶点 z，定义如下（另见图*2*）：

– z 的水平线段 $\mathbf{h}(z)$ 是从 $t(z)$ 所在点向右延伸的线段，直到到达 $t(z)$ 最右侧后代节点的 x 坐标。– z 的垂直线段 $\mathbf{v}(z)$ 是从 $b(z)$ 所在点向上延伸的垂直线段，直到到达 $t(z)$ 的 y 坐标（即与$\mathbf{h}(z)$相交）。

开发高效算法的关键在于，图中的边可以通过这些线段的交点来刻画。

定理 1. *[2]*设 G 是一个具有 *RDV* 表示的图，且 $z_1, \ldots, z_n$ 是顶点的自底向上枚举。则对于任意 $i < j$，边 (z_i, z_j) 存在，当且仅当垂直线段$\mathbf{v}(z_j)$ 与水平线段 $\mathbf{h}(z_i)$相交。

2.2 RDV 表示的预处理

对于接下来的数据结构，如果由顶点定义的每个线段集合都互不相交，将会很有帮助。我们可以通过扰动坐标来实现这一点，但我们发现，直接修改 RDV 表示更容易，而且这样还能使其具备一些可能具有独立价值的性质。下面的结论很容易证明（示例见图2）。

引理 1. 设 G 是一个具有 *RDV* 表示(T, t, b)的图。在$O(|T| + n)$ 时间内，我们可以将(T, t, b) 转换为 *RDV* 表示(T', t', b')，使得对于所有顶点对 $z \neq w$，$t'(z) \neq t'(w)$ 且 $b'(z) \neq b'(w)$ 。此外，所有底节点都是叶节点，且没有顶节点是叶节点。

证明。 假设每个节点 ν 都知道以其为顶节点的所有顶点，以及以其为底节点的所有顶点。（我们可以在 $O(|T| + n)$ 时间内计算出这些信息。）现在对每个节点 ν 执行以下操作。首先，如果存在 d 个顶点 $z_1, \ldots, z_d$，使得 ν是它们的底节点，那么就在 ν下添加 d 个新叶节点，并将它们设为 $z_1, \ldots, z_d$的新底节点。（这样一来，顶节点就不再是叶节点，因为相应的底节点变成了严格后代节点。）其次，如果存在 $k \geq 2$ 个顶点 $w_1, \ldots, w_k$，使得 ν 是它们的顶节点，那么令 ρ为 ν 的父节点（如果 ν 是根节点，则临时插入一个新节点作为 ρ 的父节点）。用 $k - 1$ 个新节点细分从 ν 到 ρ 的边，并将这些新节点设为 $w_2, \ldots, w_k$的新顶节点。容易验证，新定义的路径在某个节点处相交，当且仅当旧路径在某个节点处相交。我们为 G的每个顶点至多添加两个新节点，因此 $|T'| \leq |T| + O(n)$。

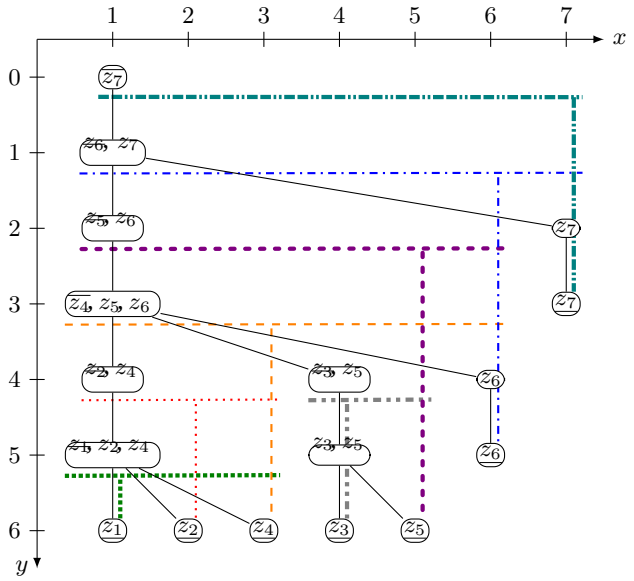

图 2. 修改后的 RDV 表示，以及与顶点对应的线段（为便于辨认，绘制时稍作偏移）。

从现在起，我们假设给定的 RDV 表示满足引理 1 的条件。由此，对于任意 $z \neq w$，$\mathbf{v}(z)$ 与 $\mathbf{v}(w)$ 不相交，因为 $b(z) \neq b(w)$ 都是叶节点，因此具有不同的 x 坐标，而这决定了垂直线段的 x 坐标。同理，$\mathbf{h}(z)$ 与

$\mathbf{h}(w)$ 不相交：如果 $t(z)$ $t(w)$ 的 y 坐标不同，那么这两条水平线段的坐标也不同。如果 $t(z)$ 和 $t(w)$ 具有相同的 y 坐标，那么它们位于宿主树的同一层级，并且根据 $t(z) \neq t(w)$，它们的后代集合不相交。由于后代的 x 坐标决定了水平线段的 x 范围，因此 $\mathbf{h}(z)$ 与$\mathbf{h}(w)$ 不相交。

3 RDV 图中的支配集

为了高效地在 RDV图中求出一个最小支配集，我们采用一种众所周知的贪心算法 [3]。假设给定了一个顶点顺序 $z_1, \ldots, z_n$。将支配集 D 初始化为空，并将所有顶点标记为“未支配”。然后，对于 $i = 1, \ldots, n$，按如下方式进行：

> 如果 z_i “未支配”，则选择顶点 $z_q \in N[z_i]$，使其在顶点顺序中的索引 q 最大，将 z_q 加入 D，并将 z_q 的所有邻居标记为“已支配”。

显然，这样得到的是一个支配集；正如 Booth 和 Johnson 所证明的那样 [3]，，如果在 RDV图上使用自底向上消去序作为顶点顺序运行该算法，就能得到一个最小支配集。此外，该算法显然可以在线性时间内实现：每当顶点 z首次变为已支配，或被加入 D时，我们花费 $O(\deg(z))$ 时间，而其余所有操作总共花费 $O(n)$ 时间。

为了将运行时间降低到边数的次线性级别（具体而言，在计算宿主树 T 的坐标后为 $O(n \log\ n)$），我们将所需的关键操作单独列出。

A. 对于顶点 z_i，找出索引 q 最大的顶点 $z_q \in N[z_i]$。B. 给定一个顶点 z_q，确保 $N[z_q]$ 中的所有顶点都标记为“已支配”。

注意，每次迭代中每项操作最多执行一次，因此总共执行 n 次，所以只需证明如何在 $O(\log n)$ 时间内实现这些操作，即可满足我们的运行时间要求。（我们只能在摊销意义上达到这一界限：所有迭代总共耗时 $O(n\log\ n)$。）

操作 A 与我们在配套论文 [2] 中需要执行的一项操作非常相似（唯一的区别是，我们使用闭邻域而非开邻域，并且求最大值而非最小值）。正如该文所示，因此可以使用优先搜索树将每次操作实现为 $O(\log n)$ 时间；我们将在下文简要回顾这一方法。

然而，这种实现只有在“当前顶点”v_i的情况下才高效，因此不能用于操作 B，因为操作 B 的输入顶点 z_q 是任意的。（我们知道 $q \geq i$，因为 $z_i \in N[z_i]$，而且我们会最大化该索引，但 $q > i$ 是有可能的。）因此，我们需要开发一种全新且截然不同的方法来支持操作 B。

3.1 在每次迭代中于 $O(\log n)$ 时间内实现操作 A

为 G 中的每个顶点 z 定义一个元组，由一个值 和一个权重组成。我们将值设为 $x(b(z))$，即$\mathbf{v}(z)$的 x坐标。我们将权重设为 z 在顶点顺序中的索引，也就是说，如果 $z = z_q$，它就是 q 。为了支持操作 A，我们将其中一些元组存储在优先搜索树 $\mathcal{P}$中。这是一种用于二维点的数据结构，由 McCreight [21]提出。它支持（除其他操作外）范围最大值查询：输入一个值范围 $[x, x']$ ，然后从存储在 $\mathcal{P}$ 中且值落在该范围内的所有元组中，返回权重最大的元组（权重相同时任意选择；若不存在这样的元组，则返回“null”）。该操作可在 $O(\log |\mathcal{P}|)$ 时间内完成 [21]。（我们的优先搜索树最多始终存储 n个元组，因此所需时间为 $O(\log n)$。）优先搜索树还支持在 $O(\log n)$ 时间内添加或删除一个元组。这样，我们就具备了操作 A 所需的所有工具。

引理 2。 我们可以在摊销 $O(\log n)$ 时间内实现操作 A。

证明。 我们维护一棵优先搜索树 $\mathcal{P}$，初始为空。我们将确保以下内容在整个过程中对 $\mathcal{P}$成立：

不变式： 在第 i次迭代中，优先搜索树 $\mathcal{P}$ 恰好存储满足以下条件的顶点 z_q 元组： $q \geq i$ 且 $y(b(z_q)) \geq y(t(z_i)) \geq y(t(z_q))$，即，$y$-范围的$\mathbf{v}(z_q)$包含 $y(t(z_i))$。

因此，在第 i次迭代开始时，我们将所有满足新近成立的 $y(b(z_p)) \geq y(t(z_i))$ 的顶点 z_p 元组加入 $\mathcal{P}$ ，这意味着 $i = 1$ 或 $y(b(z_p)) < y(t(z_{i-1}))$。（我们可以通过预先按照底部节点的 y-坐标对所有顶点建立排序列表 L，并在该坐标足够大时将顶点从L 转移到 $\mathcal{P}$ ，从而高效地找到这些顶点。）新加入 $\mathcal{P}$ 的顶点 z_p 满足 $p \geq i$，因为 $i = 1$ 或 $y(t(z_p)) \leq y(b(z_p)) < y(t(z_{i-1}))$。因此，由于我们使用自底向上消去序，$p > i-1$ 。在第 i 次迭代结束时，我们从 $\mathcal{P}$中移除 z_i 的元组，因此 $\mathcal{P}$ 中剩余的所有元组的索引都大于i ，不变式对下一次迭代成立。由于 $t(z_i(\ i\ ))$ 会随着 i 增大而减小，因此每个顶点只会被加入 $\mathcal{P}$ 一次。每个顶点也只会被删除一次，在所有迭代中维护 $\mathcal{P}$的总时间为 $O(n\log\ n)$。

回顾一下，操作 A 表示在第 i 次迭代中，找出使索引 q 最大的顶点 $z_q \in N[z_i]$。由于 $z_i \in N[z_i]$，我们有 $q \geq i$，因此只需在 $N_{\geq}[z_i]$ 中的顶点之间进行搜索。我们声称，这等价于在 $\mathcal{P}$ 中对 $\mathbf{h}(z_i)$ 的 x 范围执行范围最大值查询。首先假设范围最大值查询返回了 z_q 的元组。那么，$\mathbf{h}(z_i)$ 的 x 范围包含 $x(b(z_q))$，即 $\mathbf{v}(z_q)$ 的 x 坐标。此外，根据不变式， $q \geq i$ ，且 $\mathbf{h}(z_i)$ 的 y 坐标落在$\mathbf{v}(z_q)$ 的 y 范围内。因此，$\mathbf{v}(z_q)$ 与 $\mathbf{h}(z_i)$ 相交，且$z_q \in N_{\geq}[z_i]$。类似地可以证明，任意顶点 $z_q \in N_{\geq}[z_i]$ 都会给出 $\mathcal{P}$ 中的一个元素，其值落在该范围内。因此，范围最大值查询恰好会考察 $N_{\geq}[z_i]$ 的顶点（对应的元组），并按要求返回索引最大的那个顶点。因此，执行操作 A 就是进行一次范围最小值查询，耗时 $O(\log n)$。

在图2的示例中，第一次迭代时，我们有 $y(t(z_1)) = 5$ ，而$\mathcal{P}$存储了 $z_1, \ldots, z_6$ 的元组，因为它们的垂直线段跨越 y 坐标 5。我们搜索 x 范围 $[2, 4]$, ，而 z_1, z_2, z_4 的垂直线段落在此范围内。在这些顶点中， z_4 的权重最大，因此我们将其加入支配集。

3.2 以每次迭代 B in $O(\log n)$ 时间实现操作

在操作 B （第 i 次迭代）中，我们给定一个具有 $q \geq i$ 的顶点 z_q ，并且必须确保其闭邻域被标记为已支配。我们将其分为两个步骤，分别考虑 $N_{\leq}[z_q]$ 和 $N_{>}[z_q]$ 。如前所述，我们不能在此使用优先搜索树的思路，因为优先搜索树中存储的线段取决于 t（z_i）的 y 坐标，因此它只能用于在 $N[z_i]$内搜索。我们改为存储两种不同的数据结构（一种用于水平线段，另一种用于垂直线段），以支持射线射击查询。

首先需要一些背景知识。设 S 是一组互不相交的水平线段。射线射击查询 以一条垂直射线 $\overrightarrow{q}$为输入，该射线由其端点 q 和发出方向指定。所需的输出是 S 中被射线击中 的第一条线段，即它包含 $\overrightarrow{q}$中的某个点 p ，且开线段 qp 不与 S中的任何线段相交。（如果 $\overrightarrow{q}$没有击中 S中的任何线段，则返回“null”。）人们开发了许多数据结构，以高效回答射线查询，并支持插入或删除线段。例如，参见 [19,22]，其中介绍了处理时间较慢的早期结果。目前最佳的运行时间界由 Giyora 和 Kaplan [16]给出；他们的数据结构使用 $O(|S|)$空间，并支持在 $O(\log |S|)$时间内进行更新和查询。（在我们的情形中，我们始终有 $|S| \leq n$。）后来有研究证明，即使线段和射线彼此不正交 [24]，也能达到这些运行时间界，不过我们的应用不需要这一点。

引理 3. 我们可以在摊销 $O(\log n)$ 时间内实现操作 B。

证明。 我们先说明如何在 $N_{\leq}[z_q]$ 中标记所有未支配顶点，然后简要概述如何对称地处理 $N_{>}[z_q]$ 。在整个算法过程中，我们维护一个集合 S_H ，用于存储所有未被支配顶点的水平线段。（为了摊销分析成立，从该集合中移除已支配顶点的线段非常重要。）初始时， S_H 简单地存储所有顶点 z 的$\mathbf{h}(z)$。（请注意，这些线段互不相交。）当我们首次将 z 标记为“已支配”时，就从 S_H 中移除$\mathbf{h}(z)$。显然，这总共需要 $O(n \log n)$ 时间，因此每次迭代的摊销时间为 $O(\log n)$。

我们为每个顶点 z 定义一条射线 $\overrightarrow{z}$，它从 $\mathbf{v}(z)$ 的顶端出发并沿其方向射出（即朝 ∞ 在 y 坐标系中的方向）。假设在第 i 次迭代中，我们已固定顶点 z_q，并在由射线 $\overrightarrow{z_q}$ 构成的线段集 S_H 中执行一次射线投射查询。如果查询返回 $\mathbf{h}(z_p)$，那么我们会显式检查是否满足 $z_p \in N[z_q]$；由于可以证明 $p \leq q$ （见下文），根据定理 1，此检查只需常数时间。如果确实有 $z_p \in N_{\leq}(z_q)$，那么我们将 z_p 标记为已支配

（此前它是“未支配”的，因为它的线段位于 S_H 中）。我们从 S_H 中移除它的线段（也从我们在下文定义的“其他”数据结构 S_V 中移除）。然后重复射线查询。

为了看出必须有 $p \le q$，回想一下，我们是向下发射射线（朝着更大的 y 坐标），所以 $y(t(z_p)) \ge y(t(z_q))$. 因此只有在 $p > q$ 的情况下才可能有 $y(t(z_p)) = y(t(z_q))$，但由于顶部节点互不相同，这就意味着 $\mathbf{h}(z_p)$ 的 x 范围与$\mathbf{h}(z_q)$ 的范围不相交。后者范围包含射线的 x 坐标，因此 $\overrightarrow{z_q}$ 不可能击中 $\mathbf{h}(z_p)$。

现在假设查询返回某个顶点 z_p 对应的 $\mathbf{h}(z_p)$，其中 $p \le q$ 但 $z_p \notin N[z_q]$。我们声称，这样一来邻居搜索就完成了。（对于在 $N_{\le}[z_q]$ 中的搜索，这种情况实际上不会发生，但对于在$N_{>}[z_q]$ 中的搜索，则可能发生。）具体来说，射线与$\mathbf{h}(z_p)$ 的交点位于范围之外 $\mathbf{v}(z_q)$ 的 y 范围之外（否则根据定理 1，两者之间就会有一条边）。进一步的查询只会找到 y 坐标更大的线段，因此也会位于 $\mathbf{v}(z_q)$ 的 y 范围之外，所以不会再揭示 $N_{\le}[z_q]$ 中的任何元素。

因此，如果我们不断执行射线投射查询，直到收到“null”或找到一个不在 z_q 的邻域中的顶点，那么找到的所有顶点都在 $N_{\le}[z_q]$中。反过来， $N_{\le}[z_q]$ 中任何未支配的顶点都会被找到，因为它的水平线段与$\mathbf{v}(z_q)$相交，因此会被射线击中。以图2为例，在第一次迭代中（此时我们有 $z_q = z_4$），我们从点 (3,3) 向下发射一条垂直射线，它依次击中 z_4, z_2, z_1的水平线段，然后返回“null”。

如果我们在 $N_{\le}[z_q]$ 中找到了 k 个顶点（对于某个 $k \ge 0$），那么就必须执行$k+1$ 次射线投射查询，耗时为 $O((k+1)\log n)$。但我们也将k 个顶点标记为已支配。因此，我们可以将其中一个对数因子归于触发操作 B 的迭代 i，其余因子则归于首次被标记为“已支配”的顶点。每个顶点只会发生这种情况一次，因此总体摊销时间为 $O(\log n)$。

以对称的方式，我们可以确保 $N_{>}[z_q]$ 中的所有顶点都被标记为已支配：存储一个数据结构 S_V，其中为每个未支配顶点 z存储$\mathbf{v}(z)$。（回想一下，这些线段彼此不相交。）要查找 $N_{>}[z_q]$，从$\mathbf{h}(z_q)$的左端点向右发射一条射线，并找到它在 S_V 中相交的第一条线段。如果相交于某条线段，比如$\mathbf{v}(z_p)$，那么首先检查交点的 x坐标是否位于$\mathbf{h}(z_q)$内。如果不在，那么根据定理 1（并且由于我们返回的是被击中的第一条线段），我们就找到了 $N_{>}[z_q]$中所有未支配的顶点。如果 x坐标位于范围内，那么 z_p 是 z_q 的一个未支配邻居（并且必然在 $N_{>}[z_q]$ 中，因为我们之前已经找到了 $N_{\le}[z_q]$ 中所有未支配的顶点，并从 S_V中移除了它们的线段）。我们将其标记为已支配，并重复射线查询，直到返回“null”。在我们的例子中，我们会从点 (1,3) 向右发射一条射线。射线击中的第一条线段属于 z_5 （请记住， z_4 之前已被标记为已支配，因此其线段也从 S_V 中移除了）。我们将 z_5 标记为已支配，同样也将z_6 标记为已支配（射线接下来击中了它的线段）。接着，射线击中 z_7的垂直线段，但是

这里交点的 x坐标超出了$\mathbf{h}(z_4)$的范围；我们停止搜索，并找到了 $N_>[z_4]$中的全部顶点。

算法 1： RDV 图中的支配集

输入： 一个带有 RDV 表示 T 的图 G

计算 T 中节点的坐标。
按 $y(t(\cdot))$ 降序对顶点进行桶排序，得到顶点顺序 z_1、.....、 z_n。
按 $y(b(\cdot))$ 降序对顶点进行桶排序，得到一个列表 L。
初始化支配集 $D = \emptyset$ 和空的优先搜索树 $\mathcal{P}$
初始化正交射线查询结构 S_H 和 S_V
对于 $i = 1, 2, \cdots, n$ **do**
    S_H.插入$(\mathbf{h}(z_i))$，S_V.插入$(\mathbf{v}(z_i))$，将 z_i 标记为“未支配”。
对于 $i = 1, 2, \cdots, n$ **do**
    当 *the first vertex* z_p *in* L *satisfies* $y(b(y_p)) \geq y(t(z_i))$ **执行** // 添加到 $\mathcal{P}$
        从 L 中删除 z_p，并将其元组 $[x(b(z_q))\ q]$ 添加到 $\mathcal{P}$
    如果 z_i *is "undominated"* **那么**
        $[\cdot, q] \leftarrow \mathcal{P}$.范围最大值$(x$-$\mathbf{h}(z_i)$ 的范围$)$ // 操作 A
        $D \leftarrow D \cup \{z_q\}$
        $\overrightarrow{z_q} \leftarrow$ 从$\mathbf{v}(z_q)$ 的顶端向下射线 // 操作 B，$N_\leq$
        $s = S_H$.正交射线查询$(\overrightarrow{z_q})$
        当 s *is not null (say* $s = \mathbf{h}(z_p)$*)* **时**
            如果 $y(t(z_p))$ *lies in the y-range of* $\mathbf{v}(z_q)$，**则**
                将 z_p 标记为“已支配”，S_H.删除 $(\mathbf{h}(z_p))$，S_V.删除 $(\mathbf{v}(z_p))$
                $s \leftarrow S_H$.或射线投射$(\overrightarrow{z_q})$
            否则 $s \leftarrow$ 空
        $\overrightarrow{z_q} \leftarrow$ 从 $\mathbf{h}(z_q)$ 的左端向右的射线 // 操作 B，$N_>$
        $s = S_V$.正交射线投射$(\overrightarrow{z_q})$
        当 s *is not null (say* $s = \mathbf{v}(z_p)$*)* **时**
            如果 $x(b(z_p))$ *lies in the x-range of* $\mathbf{h}(z_q)$，**则**
                将 z_p 标记为“已支配”，S_H. 删除 $(\mathbf{h}(z_p))$，S_V. 删除 $(\mathbf{v}(z_p))$
                $s \leftarrow S_V$.正交射线投射（$\overrightarrow{z_q}$）
            否则 $s \leftarrow$ null
    从 $\mathcal{P}$删除 z_i 的元组
返回 D

至此，我们结束了对如何高效实现这些操作的描述，并得到了主要结果（我们还在算法 1 中用伪代码总结了该算法）。

定理 2. 给定一个含有 n 个顶点的图 G，其 *RDV* 表示的规模为 $O(n)$ T，可以在 $O(n \log n)$ 时间内找到 G 的最小支配集。

$O(\log n)$ 的摊销运行时间是通过 Giyora 和 Kaplan [16] 的射线查询数据结构实现的。该结构使用动态分数级联和 van Emde Boas 树的一个推广版本，因此相当复杂。另一种选择是使用 Overmars [25] 较早提出的射线射击数据结构，它简单得多。不过，该结构的更新和查询运行时间为 $O(\log^2 n)$，因此运行时间界会多出一个 $\log n$ 因子。

4 $O(n)$时间算法：区间图

如引言中所述，Cheng 等人已提出一种算法，可在运行时间为 $O(n)$ 的情况下，求出区间图的最小支配集。 [6], 但我们未能在文献中找到该算法的描述。由于运用上一节的思路，很容易给出这种算法，因此我们在此独立证明这一结果。

每个区间图都是 RDV 图，因为我们可以将其由区间相交定义的表示视为一种 RDV 表示，其中宿主树 T 是一条路径（以一端为根节点），而顶点 v 的区间则变为一条路径 $T(z)$，连接其顶端 $t(z)$与底端 $b(z)$。通过使用类似于引理 1 的操作，我们可以使得 $t(z) \neq t(w)$ 对任意两个顶点 $z \neq w$ 都成立，同时保持宿主树为大小为 $O(|T| + n)$ 的路径。由于 T是一条路径，所有 T 的节点都具有相同的 x 坐标，只有 y 坐标与交点相关。为便于描述，我们将 T 的节点与其 y 坐标（即距根节点的距离）对应起来，例如，将顶点 v的区间写作 $[t(z)\ b(z)]$，并使用诸如 $b(w) \leq t(z)$ 这样的表述。

为了在区间图中找到一个最小支配集，我们使用相同的贪心算法，但以不同的方式实现所需的两个操作。

操作 A: 我们维护一个具有相同不变式的集合 $\mathcal{P}$，但由于所有 x 坐标都相同，可以使用更简单的数据结构：我们将 $\mathcal{P}$ 存储为一个无序的顶点列表。显然，我们可以在 $O(1)$ 时间内向其中添加元素或删除元素，前提是每个顶点都记录了自己在 $\mathcal{P}$ 中的存储位置。

在迭代 i中，列表 $\mathcal{P}$ 恰好存储满足 $t(z_p) \leq t(z_i) \leq b(z_p)$的那些顶点 z_p。所有这些顶点都与 z_i 的区间相交（位于 $t(z_i)$），并且 $p \geq i$，因此 $\mathcal{P}$ 恰好存储 $N_{\geq}(z_i)$。为了找到使索引 q 最大的顶点 $z_q \in N_{\geq}(z_i)$，只需找到 $\mathcal{P}$ 中索引最大的顶点。我们通过暴力搜索在 $O(|\mathcal{P}|)$时间内完成此操作；我们将看到，这相当于 $O(1)$ 摊销时间。

我们将 z_q 加入支配集 D，并声称这样一来，z_1的所有顶点z_q，...，都已支配。（这里，图是区间图这一点至关重要。）因为如果 $1 \leq p \leq q$，那么要么 $p < i$（于是我们在更早的一次迭代中已使 z_p 已支配），要么 $t(z_q) \leq t(z_p) \leq t(z_i) \leq b(z_q)$（因为 $q \geq p \geq i$ 和 $z_q \in \mathcal{P}$），因此 z_q 和 z_p 的区间相交。

因此，操作 A（仅在 z_i 未支配时执行）之后会有 $q - i$ 次迭代，在这些迭代中将不 执行操作 A。此外，根据 q的选择，$|N_{\geq}(z_i)| \leq q - i + 1$。因此，这些“廉价”的迭代抵消了执行操作 A 的成本，从而使其摊销时间变为 $O(1)$。

操作 B： 我们必须确保 $N_<(z_q) \cup N_\geq(z_q)$中的所有顶点都标记为“已支配”。对于 $N_<(z_q)$来说，这很容易；它是 $\{z_1, \ldots, z_{q-1}\}$的子集，因为我们只需将 z_i、.....、z_{q-1} 中的所有 顶点都标记为已支配（正确性已在上文论证）。这一步所需时间为 $O(q-i)$，因此与上文一样，其摊销时间为 $O(1)$。

仍需找出在迭代 i之前尚未被支配的 $N_\geq(z_q)$中的所有顶点。注意，任何 满足 $p > q$ 的顶点 z_p 此时都未被支配：由 q的选择可知，我们有 $z_p \notin N(z_i)$；又由于 $p > q > i$， $t(z_p) < t(z_i)$，因此我们有 $t(z_p) \leq b(z_p) < t(z_i)$。另一方面，迭代 i 之前位于 D 中的任何顶点 z_d 都满足 $d < i$ （因为 z_i 尚未被支配），因此 $t(z_i) < t(z_d) \leq b(z_d)$，并且$z_p \notin N[z_d]$。所以我们只需计算 $N_\geq(z_q)$。不过，我们已经通过操作 A 看到了计算它的方法：这与迭代 q开始时 $\mathcal{P}$ 的内容相同。因此，我们不实现操作 B，而是等到迭代 q （中间的所有迭代都很简单，因为那些顶点已被 z_q支配），并在该次迭代中将对应的邻居标记为“已支配”。由于它们此前尚未被支配，这需要 $O(1)$ 摊销时间。

我们在算法 2 的伪代码中总结该算法。

定理 3。 给定一个具有 n 个顶点的区间图 G，其采用 $O(n)$大小的 *RDV* 表示 T，且宿主树为一条路径，可以在 $O(n)$时间内找到其最小支配集 G。

算法 2： 区间图中的支配集

输入： 具有 RDV 表示的图 G，其中宿主树 T 是一条路径

计算 T 的节点坐标。
按照 $y(t(\cdot))$ 的降序对顶点进行桶排序，以获得顶点顺序 z_1、.....、z_n。
按照 $y(b(\cdot))$ 的降序对顶点进行桶排序，以获得列表 L。
初始化支配集 $D = \emptyset$，并创建一个空列表 $\mathcal{P}$
将所有顶点标记为“未支配”
对于 $i = 1, 2, \cdots, n$ **do**
    当 *the first vertex* z_q *in* L *satisfies* $y(b(z_q)) \geq y(t(z_i))$ **执行**
        删除 z_q 从 L 中删除，并将其添加到 $\mathcal{P}$
    如果 z_i *is “undominated”*，**则**
        在 $\mathcal{P}$ 中查找索引最大的顶点 z_q // 操作 A
        $D \leftarrow D \cup \{z_q\}$
        对于 $j = i$ *to* $q-1$ **执行** 将 z_j 标记为“已支配” // 操作 B, $N_<$
    如果 z_i *belongs* D，**则** // z_i 在上一次操作 A 期间为 z_q
        将 $\mathcal{P}$ 中的所有顶点标记为“已支配” // 操作 B, $N_\geq$
    从 $\mathcal{P}$ 中删除 z_i
返回 D

5 进一步思考

本文给出了一种用于在 RDV图中寻找最小支配集的次线性时间算法。其核心是通过涉及优先搜索树和射线投射的数据结构，高效地实现所需的操作（在一种自然的贪心算法中）。

我们的方法也可用于其他问题的次线性时间算法。举例来说，考虑独立集问题：我们要找出一个最大的顶点集合 I，其中任意两个顶点都不相邻。在弦图 [18]中，有一种自然的贪心算法可以得出正确答案：固定一个所谓的完美消去序 $z_1, \ldots, z_n$（对于 RDV图，自底向上的消去序就是完美消去序）。将 I 初始化为空集合，并将所有顶点标记为“未覆盖”。对于 $i = 1, \ldots, n$，如果 z_i 尚未被覆盖，则将 z_i 加入 I，并将 $N[z_i]$ 中的所有顶点标记为“已覆盖”。借助操作 B，我们可以在 RDV图中以 $O(k \log n)$ 时间执行将所有邻居标记为已覆盖的操作，其中 k 是此前未被覆盖的顶点数。由于每个顶点只会经历一次这一过程，因此在 RDV图中的运行时间为 $O(n \log n)$。

我们最后提出一个开放问题。定理 2 的运行时间能否改进到 $O(n \log \log n)$，甚至 $O(n)$？一种可能的方法是改进优先搜索树和正交射线查询数据结构的运行时间，至少在所有坐标都是 $O(n)$ 中的整数这一特殊情况下。不过，也有可能将用于区间图的算法推广开来，即进一步利用 RDV 图的结构，论证更简单的数据结构便足以执行这些操作。

参考文献

1. Alan A. Bertossi 和 Allessandro Gori。加权区间图中的全支配与不可约性。 *SIAM J. Discret. Math.*，1(3):317–327，1988年。doi:10. 1137/0401032。2. T. Biedl 和 P. Gokhale。高效地求解 RDV 图中的最大匹配。*Computing in Geometry and Topology*，2025年。已投稿至 CCCG’24 论文专刊。3. K. Booth 和 J.H. Johnson。弦图中的支配集。*SIAM Journal on Computing* 11 1 191 199, 1982 10 1137 0211015 , ( ): – 。

doi: . / 。4. Andreas Brandst¨adt。反馈顶点集、哈密顿回路、支配集、斯坦纳树和带宽在特殊完美图上的计算复杂性。 *J. Inf. Process. Cybern.*，23(8/9):471–477，1987年。5. M.-S. Chang、S.-L. Peng、和 J.-L. Liaw。延迟查询：一种解决区间图中若干问题的高效方法。 *Networks*，34(1):1–10，1999年。doi:10.1002/ (SICI)1097-0037(199908)34:1。6. S. W. Cheng、M. Kaminski、和 S. Zaks。直线上区间的最小支配集。 *Algorithmica*，20(3):294–308，1998年3月。doi:10.1007/PL00009197。7. V. Chvatal。集合覆盖问题的贪心启发式算法。 *Mathematics of Operations Research* 4 3 233 235, 1979 10 1287 4 3 233 , ( ): – 。

doi: . /moor. . . .

8. D.G. Corneil 和 Y. Perl。完美图中的聚类与支配。 *Discrete Applied Mathematics* 9 1 27 39, 1984 10 1016 0166 218 84 90088 , ( ): – 。 doi: . / - X( ) -X。9. D.G. Corneil 和 L.K. Stewart。完美图中的支配集。 *Discrete Mathematics* 86 1 145 164, 1990 10 1016 0012 365 90 90357 , ( ): – 。 doi: . / - X( ) -N。10. R. Diestel。 *Graph Theory 4th Edition* 173 *Graduate texts in mathematics* , , 卷 。 2012 施普林格，。11. M. Farber。强弦图中的支配、独立支配与对偶性。 *Discrete Appl. Math*, 7(2):115–130，1984年。doi:10.1016/0166-218X(84) 90061-1。12. M. Farber 和 J.M. Keil。排列图中的支配。 *Journal of Algorithms* 6 3 309 321, 1985 10 1016 0196 6774 85 90001 , ( ): – 。 doi: . / - ( ) -X。13. M. R. Garey 和 D. S. Johnson。 *Computers and Intractability: A Guide to the Theory of NP-Completeness* 1979 。 Freeman，。14. F. Gavril。树中子树的交图恰好是弦图。 *Journal of Combinatorial Theory Ser. B*，16:47–56，1974年。doi:10.1016/0095-8956(74)90094-X。15. F. Gavril。有向树中有向路径交图的识别算法。 *Discret. Math.*, 13(3):237–249，1975年。doi:10.1016/0012-365X(75) 90021-7。16. Y. Giyora 和 H. Kaplan。直线型平面划分中的最优动态垂直射线查询。 *ACM Trans. Algorithms*, 5(3):28:1–28:51，2009年。doi:10.1145/ 1541885.1541889。17. P. Gokhale。RDV 图的高效算法。硕士论文，滑铁卢大学 David R. Cheriton 计算机科学学院，2025年。18. M. C. Golumbic。 *Algorithmic graph theory and perfect graphs*。学术出版社，纽约，第一版，1980年。19. H. Kaplan，E. Molad 和 R. Tarjan。带优先级的动态矩形相交。载于 *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing* 03, 639 648, 2003 ，S TOC ’，第 – 页，纽约，NY，美国，。计算机协会。doi:10.1145/780542.780635。20. Y. Liang 和 C. Rhee。圆弧图中的最大匹配。*Information Processing Letters*，45(4):185–190，1993年。doi:10.1016/0020-0190(93) 90117-R。21. E.M. McCreight。优先搜索树。 *SIAM Journal on Computing*，14(2):257–276，1985年。doi:10.1137/0214021。22. K. Mehlhorn 和 S. N¨aher。动态分数级联。*Algorithmica*，5 1 4 215 241，2023 10 1007 01840386 (–): – 。doi: . /BF 。23. C. Monma 和 V. Wei。树中路径的交图。 *J. Comb. Theory,* *Ser. B* 41 2 141 181, 1986 10 1016 0095 8956 86 90042 0 , ( ): – 。 doi: . / - ( ) - 。24. Y. Nekrich。最优时间内的动态平面点定位。载于 *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*，STOC 2021，第1003–1014页，2021年。doi:10.1145/3406325.3451100。25. M.H. Overmars。线段集合的范围搜索。载于 *Proceedings of theFirst Annual Symposium on Computational Geometry*，SCG ’85，第177–185页，纽约，NY，美国，1985年。计算机协会。doi:10.1145/323233.323257。26. G. Ramalingam 和 C. Pandu Rangan。区间图上支配问题的统一方法。 *Inf. Process. Lett.*，27(5):271–274，1988年。doi:10.1016/ 0020-0190(88)90091-9。

一个缺失的证明

引理 1. 设 G 是一个具有 *RDV* 表示 (T, t, b)的图。那么，我们可以在$O(|T| + n)$ 时间内，将 (T, t, b) 转换为 *RDV* 表示 (T', t', b')，使得 $t'(z) \neq t'(w)$ 且 $b'(z) \neq b'(w)$ 对所有顶点对 $z \neq w$均成立。此外，所有底节点都是叶节点，且没有顶节点是叶节点。

证明. 我们假设图 T 中的每个节点 ν 都知道所有以其为顶节点的顶点，以及所有以其为底节点的顶点。（我们可以在 $O(|T| + n)$ 时间内计算出这些信息。）现在，在图 T的每个节点 ν 处执行以下操作。首先，如果存在 d 个顶点 $z_1, \ldots, z_d$，其底节点是 ν，那么就添加 d 个新的叶节点作为 ν的子节点，并将它们设为 $z_1, \ldots, z_d$的新底节点。（这样一来，顶节点不再是叶节点，因为相应的底节点变成了严格后代节点。）其次，如果存在 $k \geq 2$ 个顶点 $w_1, \ldots, w_k$，其顶节点是 ν，那么令 ρ为 ν 的父节点（如果 ν 是根节点，则暂时插入一个新节点作为 ρ 的父节点）。用 $k - 1$ 个新节点细分从 ν 到 ρ 的边，并将它们设为 $w_2, \ldots, w_k$的新顶节点。不难验证，新定义的路径相交于一个节点，当且仅当旧路径相交于一个节点。我们为图 G中的每个顶点至多添加了两个新节点，因此 $|T'| \leq |T| + O(n)$。