# 大型图电阻距离计算的理论与实践高效方法

Yichun Yang 北京理工大学 中国北京
yc.yang@bit.edu.cn

Longlong Lin Rong-Hua Li 西南大学 北京理工大学 中国重庆 中国北京 longlonglin@swu.edu.cn lironghuabit@126.com

Meihao Liao 北京理工大学 中国北京
mhliao@bit.edu.cn

Guoren Wang 北京理工大学 中国北京
wanggrbit@gmail.com

arXiv:2601.11159v1 [cs.LG] 16 Jan 2026

**摘要**

电阻距离的计算在众多图分析应用中至关重要，包括图聚类、链接预测和图神经网络。尽管其具有基础性的重要作用，但目前仍缺乏能够在大型图上高效计算电阻距离的算法。现有的最先进水平（SOTA）方法，包括基于幂迭代的算法和基于随机游走的局部方法，通常面临收敛速度缓慢的问题，尤其是在图拉普拉斯矩阵的条件数（记为 $\kappa$）较大时。为应对这一挑战，我们受经典 Lanczos 方法启发，提出了两种新颖且高效的算法：Lanczos 迭代和 Lanczos Push，二者均旨在降低对 $\kappa$ 的依赖。其中，Lanczos 迭代是一种近线性时间的全局算法，而 Lanczos Push 是一种时间复杂度与图规模无关的局部算法。具体而言，我们证明了 Lanczos 迭代的时间复杂度为 $\tilde{O}(\sqrt{\kappa}m)$（其中，$m$ 表示图的边数，$\tilde{O}$ 表示省略对数项后的复杂度），与此前基于幂迭代的全局方法相比，实现了 $\sqrt{\kappa}$ 的加速。对于 Lanczos Push，我们证明，在某些温和且经常成立的假设下，其时间复杂度为 $\tilde{O}(\kappa^{2.75})$，相比最先进水平的基于随机游走的局部算法，$\kappa^{0.25}$ 得到了显著改进。我们在八个规模和统计特性各异的真实世界数据集上开展了广泛实验，验证了所提出的算法，结果表明，无论在效率还是准确性方面，Lanczos 迭代和 Lanczos Push 均显著优于最先进方法。

**PVLDB 引用格式：**
Yichun Yang、Longlong Lin、Rong-Hua Li、Meihao Liao 和 Guoren Wang。《大型图上电阻距离计算的理论与实践高效方法》。PVLDB，19(1): XXX-XXX，2026。doi:XX.XX/XXX.XX

**PVLDB 工件可用性：**

源代码、数据和/或其他工件已发布于 https://github.com/Ychun-yang/LanczosPush。

本作品采用知识共享署名-非商业性使用-禁止演绎 4.0 国际许可协议授权。请访问 https://creativecommons.org/licenses/by-nc-nd/4.0/ 查看该许可协议的副本。如需进行该许可协议未涵盖的任何用途，请发送电子邮件至 info@vldb.org 获取许可。版权归所有者/作者所有。出版权已授权给 VLDB 基金会。VLDB 基金会会刊，第 19 卷，第 1 期，ISSN 2150-8097。doi:XX.XX/XXX.XX

## 1 引言

电阻距离（RD）计算是图数据管理中的一项基础任务，可有效量化图中节点之间的相似性。给定一个无向图 G，两个顶点 $s$和 $t$ 之间的电阻距离 $r_{\mathcal{G}}(s, t)$ 与从 $s$出发、经过 $t$后再返回 $s$的随机游走的预期步数成正比。因此，较小的 $r_{\mathcal{G}}(s, t)$ 表明 $s$和 $t$ [20, 49]之间具有较高的相似性。RD 能有效量化节点相似性，是一种用途广泛且功能强大的工具，在图聚类 [37],链接预测 [36],图稀疏化 [40],最大流计算 [25, 42], [4, 和 53] 图神经网络等众多现实应用中发挥着作用。因此，设计快速的 RD 计算算法是一个自然的问题，近年来已受到广泛关注 [14, 19–21, 33, 49, 51]。

然而，精确计算 $r_{\mathcal{G}}(s, t)$ 等价于求解一个线性系统。目前已知只能通过复矩阵乘法 [47], 在 $O(n^{2.37})$ 时间内求解，这对于超大规模图而言效率低下。因此，近期提出了许多近似算法，用于估算 $r_{\mathcal{G}}(s, t)$，并保证绝对误差不超过 [20, 21, 33, 49]（详见定义 2.2），以平衡效率与质量。现有的近似方法大致可分为两类：（i）时间复杂度近乎线性的算法；（ii）不依赖图规模的局部算法。例如，Cohen 等人 [11] 开发了一种全局拉普拉斯求解器，可在 $O(m \log^{1/2} n \log \frac{1}{\epsilon})$ 时间复杂度内实现 a $\epsilon$-绝对误差近似，其中 $n$（分别为 $m$）表示图的顶点（分别为边）数量，$\epsilon$为绝对误差参数。然而，尽管理论上拉普拉斯求解器功能强大，但由于复杂的数据结构以及复杂度中隐藏的较大常数项和对数项，它在实际应用中处理真实世界数据集时仍然效率低下。因此，提出了更简单的幂法 [33, 34, 49]，该方法可实现 $\epsilon$-绝对误差近似，且比拉普拉斯求解器更易于实现。具体而言，幂法 [33, 34, 49]的时间复杂度为 $O(\kappa m \log \frac{\kappa}{\epsilon})$，其中 $\kappa$为归一化拉普拉斯矩阵 L 的条件数（详见第 2 节）。然而，由于时间复杂度与 $\kappa$存在固有的线性依赖关系，幂法对于条件数较大的图 $\kappa$仍然计算成本高昂

复杂度。即使在 $\kappa = O(1)$ 的最优情况下（即扩展图 [14]），幂法的时间复杂度也会降至 $O(m \log \frac{1}{\epsilon})$。然而，这种降低仍需执行 $O(\log \frac{1}{\epsilon})$ 次矩阵-向量乘法运算，因此对于超大规模图而言效率低下，因为每次此类运算都需要遍历整个图。

为进一步提高效率，研究人员最近开始采用局部算法[1] 来估计 RD [33]。这些算法只检查局部子图，而不要求访问完整图，从而显著降低了计算开销。非正式地说，我们考虑众所周知的邻接模型 [34]，该模型支持在常数时间内对图 G 进行以下三类查询：(i) 度查询；(ii) 邻居查询；(iii) 跳转查询。局部算法旨在通过尽可能少的查询，在不搜索整个图的情况下近似 $r_{\mathcal{G}}(s,t)$。例如，Peng 等人 [33] 提出了用于局部计算 $r_{\mathcal{G}}(s,t)$ 的随机游走算法。Yang 等人 [49] 进一步将幂法与随机游走相结合，改进了这一结果。然而，这些局部算法在估计 $\epsilon$-绝对误差近似 $r_{\mathcal{G}}(s,t)$ 时的时间复杂度为 $\tilde{O}(\kappa^3/\epsilon^2)$[2]。这些结果表明，尽管局部算法的复杂度不依赖于 $n$ 和 $m$，但其代价是对条件数 $\kappa$ 和参数 $\epsilon$ 的依赖更高。不幸的是，真实世界图中的条件数 $\kappa$ 可能非常大（表 2）。这使得 $\kappa^3$ 界非常高。正如我们的实验所示，我们还发现，现有的 RD 计算算法只有在 $\kappa$ 相对较小的数据集（例如社交网络）上才能高效实现，而它们在 $\kappa$ 相对较大的数据集（例如道路网络）上的性能并不理想。

因此，一个自然而重要的问题随之提出：我们能否设计出高效的线性时间和次线性时间（局部）电阻距离计算算法，同时减弱对条件数 $\kappa$ 的依赖？本文对这一问题给出了肯定的回答。首先，我们利用经典的 Lanczos 迭代算法设计了一个全局电阻距离算法。具体而言，Lanczos 迭代算法（详见第 3 节）以 $k = \sqrt{\kappa} \log \frac{\kappa}{\epsilon}$ 次搜索得到一系列正交向量 $\mathrm{v}1, ..., \mathrm{v}_k$，它们位于 $k$ 维 Krylov 子空间 $\mathrm{K}_k(\mathrm{A}, \mathrm{v}_1) = span\mathrm{v}_1, \mathrm{A}\mathrm{v}1, ..., \mathrm{A}^k\mathrm{v}_1$中，其中 $\kappa$ 表示归一化拉普拉斯矩阵 L 的条件数。我们证明，当取 $\mathrm{v}_1 = \left(\mathrm{e}_{\sqrt{d_s}}^{s} - \mathrm{e}_{\sqrt{d_t}}^{t}\right) / \sqrt{\frac{1}{d_s} + \frac{1}{d_t}}$时，利用这些正交向量 $\mathrm{v}1, ..., \mathrm{v}_k$，即可对 $r_{\mathcal{G}}(s,t)$ 进行近似，并保证其绝对误差不超过 $\epsilon$；其中 $d_s, d_t$ 表示两个指定节点 $s, t$的度。接下来，通过实现Lanczos 迭代，使用 $k$次矩阵-向量查询即可计算出 $\mathrm{v}1, ..., \mathrm{v}_k$。因此，这为电阻距离估计带来了一个时间复杂度为 $O(\sqrt{\kappa} m \log \frac{\kappa}{\epsilon})$ 的算法，相比幂法减少了对 $\sqrt{\kappa}$ 的依赖。随后，为了得到复杂度低于 $\kappa^3$ 的局部算法，我们引入了局部Lanczos 递推这一新概念（详见公式（ 3 ））。利用这些新技术，我们设计了 Lanczos Push 算法（详见第 4 节），该算法计算出一系列稀疏向量 $\hat{}\mathrm{v}1, ..., \hat{}\mathrm{v}_k$，它们与 Lanczos 迭代计算得到的正交向量 $\mathrm{v}1, ..., \mathrm{v}_k$ 接近。因此，我们的 Lanczos Push 算法估计 $r_{\mathcal{G}}(s,t)$，方法是只搜索图中的一小部分，同时仍保留 Lanczos 迭代带来的加速效果。在若干温和且常见的假设下（这些假设也已在我们的实验中得到验证），我们从理论上证明，Lanczos Push 算法的时间复杂度为 $\tilde{O}(\kappa^{2.75} C_1 C_2 / \epsilon)$，能够得到电阻距离值的绝对误差近似，其关于 $\kappa$为次立方级。关于 $C_1$ 和 $C_2$这两个数的详情，请参见表 1 和第 4 节。另一方面，我们证明，对于任何以绝对误差 $\epsilon \leq 0.01$近似 $r_{\mathcal{G}}(s, t)$ 的算法，其时间复杂度下界为 $\Omega(\kappa)$。这表明，局部算法的时间复杂度确实与条件数 $\kappa$有关，而此前关于局部电阻距离计算 [3]的研究并未强调这一点。

我们进行了大量实验，以评估所提出方案的有效性和可扩展性。实验结果表明，所提出的 Lanczos 迭代和 Lanczos Push 算法的性能显著优于所有基线方法。具体而言，Lanczos 迭代在社交网络上的速度比幂法快 5×，在道路网络上的速度比幂法快 100×。Lanczos Push 在社交网络上的速度比所有最先进的算法快 5× 至 10×，在道路网络上的速度比所有其他算法（包括 Lanczos 迭代）快 50×。总之，我们的算法在理论和实践上都高效。

## 2 预备知识

### 2.1 符号与概念

考虑一个无向连通图 $\mathrm{G} = (\mathrm{V}, \mathrm{E})$，其中有 $|\mathrm{V}| = n$ 个顶点和 $|\mathrm{E}| = m$ 条边。本文主要研究无权图。不过，所有算法和分析都可以直接推广到加权图。对于任意节点 $u \in \mathrm{V}$，其邻域定义为 $\mathrm{N}(u) = \{v | (u, v) \in \mathrm{E}\}$，其度为 $d_u = |\mathrm{N}(u)|$。度矩阵 D 是一个 $n \times n$对角矩阵，其元素为 $\mathrm{D}_{i,i} = d_i$；邻接矩阵 A 满足：若 $(i, j) \in \mathrm{E}$，则 $\mathrm{A}_{i,j} = 1$；否则 $\mathrm{A}_{i,j} = 0$。拉普拉斯矩阵定义为 $\mathrm{L} = \mathrm{D} - \mathrm{A}$，其特征值为 $0 = \lambda_1 < \lambda_2 \leq \lambda_3 \leq ... \leq \lambda_n$，对应的特征向量为 $\mathrm{u}1, ..., \mathrm{u}_n$。拉普拉斯伪逆定义为 $\mathrm{L}^\dagger = \sum_{i=2}^{n} \frac{1}{\lambda_i} \mathrm{u}_i \mathrm{u}_i^T$。根据 $\mathrm{L}^\dagger$ 的定义，电阻距离（RD）表示如下。

定义 *2.1*。给定图 G 中的一对顶点 $s$ 和 $t$，$s$和 $t$的电阻距离定义为 $r_{\mathcal{G}}(s, t) = (\mathbf{e}_s - \mathbf{e}_t)^T \mathbf{L}^\dagger (\mathbf{e}_s - \mathbf{e}_t)$。其中，$\mathbf{e}_s$（分别为 $\mathbf{e}_t$）是独热向量，在 $s$（分别为 $t$）处取值为 1，在其他位置取值为 0。

由于通过复矩阵乘法计算精确的电阻距离需要 $O(n^{2.37})$ 时间，对于大型图而言，计算代价过高。因此，现有研究主要致力于开发具有可证明误差保证的近似算法，以平衡效率与质量。本文沿用 [20, 33, 49], 的做法，同样关注具有绝对误差保证的电阻距离。

定义 *2.2*。给定图 G 和一对顶点 $s$ 与 $t$，对于任意小常数 $\epsilon > 0$，若 $|r_{\mathcal{G}}(s,t) - \hat{} r_{\mathcal{G}}(s,t)| \leq \epsilon$，则称 $\hat{} r_{\mathcal{G}}(s,t)$ 是 $\epsilon$-绝对误差近似 $r_{\mathcal{G}}(s,t)$。

[1]本文中，“局部”是指算法的运行时间与图规模 $m$ 无关或呈次线性相关。[2]$\tilde{O}(.)$ 表示省略 $n, m$ 和 $\epsilon$ 中的 log 项后的复杂度。

[3]$\kappa$的依赖关系对应的上界和下界之间仍存在较大差距，因此我们将如何使上界和下界相匹配留作未来工作。

**表 1：使用 $\epsilon$ 绝对误差保证计算成对RD的不同算法比较。对于 Lanczos Push，其中 $C_1 = \max_{u=s,\ t;i\le k} \|\mathbf{D}^{1/2}T_i(\mathbf{P})\mathbf{e}_u\|_1$，且 $C_2 = \max_{i\le k}(\|\mathbf{v}_i\|_1 + \|\mathcal{P}\mathbf{v}_i^+\|_1 + \|\mathcal{P}\mathbf{v}_i^-\|_1)$，详见第4节。**

| 方法 | 算法 | 时间复杂度 |
|---|---|---|
| 线性 | 幂法 [49] | $O(\kappa m$ 对数$-\frac{\kappa}{\epsilon})$ |
| | LapSolver [35] | $O(m$ 对数$^{1/2}\, n$ 对数$-\frac{1}{\epsilon})$ |
| | FastRD [24] | $\tilde{O}(m/\epsilon^2)$ |
| | Lanczos（第3节） | $O(\sqrt{\kappa m}\log\frac{\kappa}{\epsilon})$ |
| 次线性（局部） | TP [33] | $\tilde{O}(\kappa^4/\epsilon^2)$ |
| | TPC [33] | $\tilde{O}(\kappa^3/\epsilon^2)$ |
| | GEER [49] | $\tilde{O}(\kappa^3/(\epsilon^2 d^2))$ |
| | 算法 1 位于 [51] | $\tilde{O}(\kappa^3/(\epsilon\sqrt{d}))$ |
| | BiSPER [13] | $\tilde{O}(\kappa^{7/3}/\epsilon^{2/3})$ |
| | Lanczos Push（第4节） | $\tilde{O}(\kappa^{2.75}C_1C_2/\epsilon\sqrt{d})$ |
| 其他 | Push [20] | × |
| | BiPush [20] | × |
| | 下界（第5节） | $\Omega(\kappa)$ |

## 2.2 现有方法及其缺陷

在分析中，我们将归一化邻接矩阵定义为 $\mathbf{A} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$，并将归一化拉普拉斯矩阵定义为 $\mathbf{L} = \mathbf{I} - \mathbf{A}$。$\mathbf{L}$ 的条件数定义如下 [14]。

定义 *2.3.* 归一化拉普拉斯矩阵 $\mathbf{L}$ 的条件数 $\kappa$ 定义为 $\kappa \triangleq \frac{2}{\mu_2}$，其中 $\mu_2$ 是 $\mathbf{L}$ 的第二小特征值，即$\mathbf{L}$ 最小的非零特征值。

大多数现有 RD 算法的时间复杂度与 $\kappa$有关。具体而言，令 $\mathbf{P} \triangleq \mathbf{A}\mathbf{D}^{-1}$ 为概率转移矩阵，这些现有算法基于以下泰勒展开公式 [33, 51]。

$$r_{\mathcal{G}}(s,t) = (\mathbf{e}_s - \mathbf{e}_t)^T\mathbf{L}^{\dagger}(\mathbf{e}_s - \mathbf{e}_t) = \frac{1}{2}(\mathbf{e}_s - \mathbf{e}_t)^T \sum_{k=0}^{+\infty}\mathbf{D}^{-1}\left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)^k(\mathbf{e}_s - \mathbf{e}_t). \quad (1)$$

因此，我们可以设置足够大的截断步数 $l$，并定义近似 $r_{\mathcal{G}}(s,t) = \frac{1}{2}(\mathbf{e}_s - \mathbf{e}_t)^T \sum_{k=0}^{l}\mathbf{D}^{-1}\left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)^k(\mathbf{e}_s - \mathbf{e}_t)$。为了使 $r_{\mathcal{G}}(s,t)$ 成为 $\epsilon$-绝对误差近似 $r_{\mathcal{G}}(s,t)$，截断步数 $l = O(\kappa\log\frac{\kappa}{\epsilon})$ [33, 49, 51]。基于这一解释，可以很容易地推导出基本幂法和基于随机游走的方法。表 1 总结了最先进的方法和我们的解决方案。

幂法（PM）是数值线性代数中最经典、最基础的算法之一。PM 也广泛用于 PageRank [31], Heat Kernal PageRank [50], SimRank [43], 和图传播 [43]的计算。在我们的成对 RD 计算问题中，我们对 PM 的实现进行了轻微修改，并使用它来计算真实 RD 值。伪代码说明见算法 1。具体而言，PM 算法实现如下：（i）首先，我们定义近似 $r_{\mathcal{G}}(s,t) = 0$ 以及一个向量 $\mathbf{r} = \mathbf{e}_s - \mathbf{e}_t$，迭代次数为 $k = 0$；（ii）对于算法的第 $k^{th}$ 次迭代，给定 $k \le l$，我们执行 $r_{\mathcal{G}}(s,t) \leftarrow r_{\mathcal{G}}(s,t) + \mathbf{r}(s)/2d_s - \mathbf{r}(t)/2d_t$，并令 $\mathbf{r} \leftarrow \left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)\mathbf{r}$，然后转到（$k+1$）$^{th}$次迭代；（iii）经过 $l$次迭代后，我们停止算法并输出 $r_{\mathcal{G}}(s,t)$。根据公式 (1)，我们很容易看出，经过这一过程后输出为 $r_{\mathcal{G}}(s,t) = \frac{1}{2}(\mathbf{e}_s - \mathbf{e}_t)^T \sum_{k=0}^{l}\mathbf{D}^{-1}\left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)^k(\mathbf{e}_s - \mathbf{e}_t)$。接下来，我们证明，当设置 $l = O(\kappa\log\frac{\kappa}{\epsilon})$ 时，近似 $r_{\mathcal{G}}(s,t)$ 是 $r_{\mathcal{G}}(s,t)$ 的 $\epsilon$-绝对误差近似。由于篇幅限制，本文中省略的证明可以在完整版 [52] 中找到。

**算法 1：** 用于 RD 计算的幂法（PM）

**输入：** $\mathcal{G}$ $l$, $s$, $t$

$r_{\mathcal{G}}(s,t) = 0$, $\mathbf{r} = \mathbf{e}_s - \mathbf{e}_t$;

**对于** $i = 0, 1, 2, \ldots, l$ **do**

  $r_{\mathcal{G}}(s,t) \leftarrow r_{\mathcal{G}}(s,t) + \mathbf{r}(s)/2d_s - \mathbf{r}(t)/2d_t$;

  $\mathbf{r} \leftarrow \left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)\mathbf{r}$;

**结束**

**输出：** $r_{\mathcal{G}}(s,t)$ 作为 $r_{\mathcal{G}}(s,t)$ 的近似值

定理 2.4。 当设置 $l = 2\kappa\log\frac{\kappa}{\epsilon}$ 时，近似误差满足 $|r_{\mathcal{G}}(s,t) - r_{\mathcal{G}}(s,t)| \le \epsilon$。此外，幂法 的时间复杂度为 $O(lm) = O(\kappa m\log\frac{\kappa}{\epsilon})$。

随机游走（RW）也是各种图算法中的基本算子，包括 PageRank [22, 23, 46, 48], 热核 PageRank [10, 50] 和局部拉普拉斯求解器 [2]。对于成对RD估计，随机游走算子最早在 [33] 中被引入。该算法的总体思想如下：根据公式（1），为了近似 $r_{\mathcal{G}}(s,t)$，我们只需分别近似 $\mathbf{e}_s\left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)^i\mathbf{e}_t/d_s$、$\mathbf{e}_s\left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)^i\mathbf{e}_s/d_s$、$\mathbf{e}_t\left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)^i\mathbf{e}_t/d_t$和 $\mathbf{e}_t\left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)^i\mathbf{e}_t/d_t$，其中每个 $i \le l$ 的值恰好是从 $t$ 出发并在 $s$ 结束的长度为 $i$ 的懒惰随机游走的概率。具体而言，长度为 $i$ 的懒惰随机游走表示一个序列 $(v_0, v_1, \ldots, v_i)$，使得对于每个 $0 \le j < i$，以 1/2 的概率，$v_{j+1}$ 是 $v_j$ 的随机邻居，并以 1/2 的概率为 $v_{j+1} = v_j$。伪代码的详细内容见算法 2。通过将随机游走次数设置为 $n_r = O(l^2/\epsilon^2)$，并令 $l = O(\kappa\log\frac{\kappa}{\epsilon})$ [33],，我们可以获得具有 $\epsilon$-绝对误差近似的 $\hat{r}_{\mathcal{G}}(s,t)$。因此，该算法的总时间复杂度为 $O(n_r l^2) = \tilde{O}(\kappa^4/\epsilon^2)$，因为 $l = O(\kappa\log\frac{\kappa}{\epsilon})$ 且 $n_r = O(l^2/\epsilon^2)$。Peng 等人还提出了一种用于估计 $\mathbf{e}_s\left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)^i\mathbf{e}_t/d_s$ 的改进算法（详见 [33] 中的算法 2）。该改进算法的时间复杂度为 $\tilde{O}(\kappa^3/\epsilon^2)$，从而降低了对 $\kappa$ 的依赖。随后，Yang 等人通过对方差进行更精确的分析，进一步改进了该算法。他们在相同的误差保证下，推导出了更精确的时间复杂度 $\tilde{O}(\kappa^3/(\epsilon^2 d^2))$，其中 $d = \min\{d_s, d_t\}$。然而，基于随机游走的算法都没有突破 $\kappa^3$ 瓶颈。

**其他方法**。除了上述两种经典方法外，还有一些基于不同技术和数据结构的相关研究。例如，近线性时间拉普拉斯求解器 [15, 18] 使我们能够在 $O(m\log^{1/2} n\log\frac{1}{\epsilon})$ 时间内计算 RD 的 $\epsilon$-近似，且不受条件数 $\kappa$影响。利用拉普拉斯求解器，还可以设计 $\tilde{O}(m/\epsilon^2)$ 电阻距离草图，并在 $\tilde{O}(\epsilon^{-2})$ 时间内查询任意一对节点的 RD [24, 40]。然而，拉普拉斯求解器在实践中难以实现，且其时间复杂度线性依赖于 $m$。此外，Liao 等人 [20] 提出了 RD 的一种新颖地标解释，并设计了若干

**算法 2：** 用于 RD 计算的随机游走（RW） [33]

**输入：** $\mathcal{G}\ \epsilon,\ l,\ n_r,\ s,\ t$

**对于** $i = 0, 1, 2, ..., l$ **do**

  从 $n_r$ 开始执行 $i$ 次相互独立的、长度为 $i$ 的惰性随机游走

  在 $s$处，令 $X_{i,\ s}$（分别为 $X_{i,\ t}$）表示在 $s$结束的游走次数

  （分别为 $t$）；

  执行 $n_r$ 次从 $i$ 开始、长度为 $i$ 的独立惰性随机游走

  在 $t$处，令 $Y_{i,s}$（分别为 $Y_{i,t}$）表示以 $s$为终点的游走次数

  （分别为 $t$）；

  $\hat{r}_{\mathcal{G}}(s,t) += \frac{X_{i_{2n_r d_s}}}{2n_r d_s} - \frac{X_{i,t}}{2n_r d_t} + \frac{Y_{i,2n_r d_t}}{2n_r d_t} - \frac{Y_{i_{2n_r d_s}}}{2n_r d_s}$；

**结束输出：** $\hat{r}_{\mathcal{G}}(s,\ t)$ 作为 $r_{\mathcal{G}}(s,\ t)$ 的近似

基于该解释、高效计算 RD 的随机游走算法。然而，这些算法属于启发式方法，缺乏基于 $n$，$m$，$\kappa$的理论保证。最近，[13, 51]采用双向技术提高了随机游走在单对 RD 计算中的效率。具体而言，Yang 等人 [51]提出了一种满足 $\epsilon$-绝对误差保证的 $\tilde{O}(\kappa^3/(\epsilon\sqrt{d}))$算法，以及一种满足相对误差保证的$\tilde{O}(\kappa^3\sqrt{d}/\epsilon)$算法。与本文工作同期，Cui 等人 [13]提出了一种满足 $\epsilon$-绝对误差保证的 $\tilde{O}(\kappa^{7/3}/\epsilon^{2/3})$算法。补充说明一点：尽管 BiSPER[13] 在渐近复杂度方面具有理论优势，但在条件数较大的图（例如道路网络）上，我们的 Lanczos Push 算法在实践中表现更优。原因可能在于，双向方法 [13, 51] 的基本框架是幂迭代法，通常需要 $O(\kappa)$次迭代才能收敛；而 Lanczos Push 算法的基本框架是 Lanczos 迭代，在实践中通常只需不到 $O(\sqrt{\kappa})$次迭代即可收敛。Lanczos 迭代更强的收敛性使所提出的 Lanczos Push 算法在实践中表现更佳（如实验所示），尽管其最坏情况下的理论保证略弱。

## 3 一种新颖的 Lanczos 迭代方法

我们首先介绍用于计算二次型 $\mathbf{x}^T f(\mathbf{A})\mathbf{x}$ 的经典 Lanczos 算法，其中 $\mathbf{x} \in \mathbb{R}^n$ 是一个向量，$f$ 是一个矩阵函数，$\mathbf{A} \in \mathbb{R}^{n\times n}$ 是一个半正定（PSD）矩阵。在我们的电阻距离计算中，选择 $\mathbf{x} = \mathbf{e}_{\sqrt{d_s}}^{s} - \mathbf{e}_{\sqrt{d_t}}^{t}$，令 $\mathbf{A} = \mathbf{A}$ 为归一化邻接矩阵，并令 $f(\mathbf{A}) = (\mathbf{I} - \mathbf{A})^{\dagger}$。根据电阻距离的定义，我们有：

$$\begin{aligned} r_{\mathcal{G}}(s,t) &= (\mathbf{e}_s - \mathbf{e}_t)^T \mathbf{L}^{\dagger}(\mathbf{e}_s - \mathbf{e}_t) \\ &= (\mathbf{e}_s - \mathbf{e}_t)^T \mathbf{D}^{-1/2}(\mathbf{I} - \mathcal{A})^{\dagger}\mathbf{D}^{-1/2}(\mathbf{e}_s - \mathbf{e}_t) \\ &= (\frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}})^T (\mathbf{I} - \mathcal{A})^{\dagger}(\frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}}) \\ &= \mathbf{x}^T f(\mathcal{A})\mathbf{x}. \end{aligned} \tag{2}$$

第二个等式成立是因为$\mathbf{L}^{\dagger} = \mathbf{D}^{-1/2}\mathbf{L}^{\dagger}\mathbf{D}^{-1/2} = \mathbf{D}^{-1/2}(\mathbf{I}-\mathbf{A})^{\dagger}\mathbf{D}^{-1/2}$。这使我们可以立即使用 Lanczos 迭代来计算电阻距离值。不过，我们注意到，$f(\mathbf{A}) = (\mathbf{I} - \mathbf{A})^{\dagger}$是矩阵$\mathbf{I} - \mathbf{A}$ 的伪逆，而这并不是常见的矩阵函数（例如矩阵逆、矩阵指数）。因此，我们在算法 3 中给出 Lanczos 算法的一种实现 [29]，并做了少许修改。Lanczos 的主要思想是迭代是将矩阵 A 投影到三对角矩阵 T 上，使得 $\mathbf{T} = \mathbf{V}^T\mathbf{A}\mathbf{V}$，其中 $\mathbf{V} = [\mathbf{v}1, ..., \mathbf{v}_k]$ 是一个 $n \times k$ 维列正交矩阵，其 $k \ll n$ 张成 $k$维克雷洛夫子空间：$\mathbf{K}_k(\mathbf{v}_1, \mathbf{A}) = span\mathbf{v}_1, \mathbf{A}\mathbf{v}1, ..., \mathbf{A}^k\mathbf{v}_1$，其中 $\mathbf{v}_1 = \mathbf{x}/\|\mathbf{x}\|_2 = \left(\mathbf{e}_{\sqrt{d_s}}^{s} - \mathbf{e}_{\sqrt{d_t}}^{t}\right)/\sqrt{\frac{1}{d_s} + \frac{1}{d_t}}$。首先，若我们已经计算出列正交矩阵 V，则证明对于向量 $\mathbf{A}^i\mathbf{v}_1\ i \le k$，有 $\mathbf{A}^i\mathbf{v}_1 = \mathbf{V}\mathbf{T}^i\mathbf{V}^T\mathbf{v}_1$。形式化地，我们陈述如下引理。

引理 *3.1*。给定 $V = [v1, ..., v_k]$ 是张成子空间 $K_k(v_1,A) = span\langle v_1, Av1, ..., A^k v_1\rangle$ 的 $n \times k$ 维正交矩阵，且 $T = V^T AV$。那么对于任意次数至多为 $k$ 的多项式 $p_k$，有 $p_k(A)v_1 = Vp_k(T)V^T v_1$。

接下来，基于引理 3.1，我们证明$\mathbf{v}_1^T\mathbf{V}(\mathbf{I} - \mathbf{T})^{-1}\mathbf{V}^T\mathbf{v}_1$ 是二次型$\mathbf{v}_1^T(\mathbf{I} - \mathbf{A})^{\dagger}\mathbf{v}_1$的良好估计量。证明基于经典分析方法，但我们对证明做了少许修改。

引理 3.2. 设$\mathbf{V} \in \mathbb{R}^{n\times k}$ 和$\mathbf{T} \in \mathbb{R}^{k\times k}$ 如引理 *3.1* 中所定义。设$\mathcal{P}_k$ 为所有度为 $\le k$ 的多项式构成的集合。那么，我们有：

$$\begin{aligned} &|\mathbf{v}_1^T\mathbf{V}(\mathbf{I} - \mathbf{T})^{-1}\mathbf{V}^T\mathbf{v}_1 - \mathbf{v}_1^T(\mathbf{I} - \mathcal{A})^{\dagger}\mathbf{v}_1| \\ &\quad \le 2\min_{p\in\mathcal{P}_k}\max_{x\in[\lambda_{min}(\mathcal{A}),\lambda_2(\mathcal{A})]}|1/(1-x) - p(x)|, \end{aligned}$$

其中，$\lambda_{min}(\mathcal{A})$和 $\lambda_2(\mathcal{A})$ 分别表示$\mathcal{A}$的最小特征值和第二大特征值。

基于引理 3.2，我们现在可以用以下公式近似计算电阻距离：

$$\begin{aligned} r_{\mathcal{G}}(s,t) &= (\frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}})^T(\mathbf{I} - \mathcal{A})^{\dagger}(\frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}}) \\ &= \left(\frac{1}{d_s} + \frac{1}{d_t}\right)\mathbf{v}_1^T(\mathbf{I} - \mathcal{A})^{\dagger}\mathbf{v}_1 \\ &\approx \left(\frac{1}{d_s} + \frac{1}{d_t}\right)\mathbf{v}_1^T\mathbf{V}(\mathbf{I} - \mathbf{T})^{-1}\mathbf{V}^T\mathbf{v}_1 \\ &= \left(\frac{1}{d_s} + \frac{1}{d_t}\right)\mathbf{e}_1^T(\mathbf{I} - \mathbf{T})^{-1}\mathbf{e}_1. \end{aligned} \tag{3}$$

第二个等式成立，因为 $\mathbf{v}_1 = \left(\mathbf{e}_{\sqrt{d_s}}^{s} - \mathbf{e}_{\sqrt{d_t}}^{t}\right)/\left\|\mathbf{e}_{\sqrt{d_s}}^{s} - \mathbf{e}_{\sqrt{d_t}}^{t}\right\|$和 $\left\|\mathbf{e}_{\sqrt{d_s}}^{s} - \mathbf{e}_{\sqrt{d_t}}^{t}\right\|_2^2 = \frac{1}{d_s} + \frac{1}{d_t}$。最后一个等式成立，因为 $\mathbf{v}_1^T\mathbf{V} = \mathbf{e}_1^T$，这是由于 $\mathbf{v}_1$ 与 v2、...、$\mathbf{v}_k$正交。现在，由于 $\mathbf{I} - \mathbf{T}$ 只是一个 $k \times k$ 维矩阵，且 $k \ll n$，我们可以通过快速矩阵乘法 [47]，仅用 $O(k^{2.37})$ 时间高效地计算$\left(\frac{1}{d_s} + \frac{1}{d_t}\right)\mathbf{e}_1^T(\mathbf{I} - \mathbf{T})^{-1}\mathbf{e}_1$。最后，剩下的就是如何高效地计算这些正交向量 v1，...，$\mathbf{v}_k$。为此，在每次迭代中，我们执行 Lanczos 递推（即算法 3 第 3–7 行）：

$$\beta_{i+1}\mathbf{v}_{i+1} = (\mathcal{A} - \alpha_i)\mathbf{v}_i - \beta_i\mathbf{v}_{i-1}, \tag{4}$$

其中 $\alpha_i$ 和 $\beta_{i+1}$ 是算法 3 第 4 行和第 6 行计算得到的标量。为了进行分析，我们给出 Lanczos 迭代输出的一个经典结果（例如，参见 [29]中的断言4.1）。我们将在后续分析中使用该结果。

命题 3.3。（输出保证）*Lanczos* 迭代（算法 *3*）计算出 $\mathbf{V} \in \mathbb{R}^{n\times k}$、$\mathbf{T} \in \mathbb{R}^{k\times k}$ 和一个附加向量$\mathbf{v}_{k+1}$，以及一个标量 $\beta_{k+1}$，使得：

$$\mathbf{A}\mathbf{V} = \mathbf{V}\mathbf{T} + \beta_{k+1}\mathbf{v}_{k+1}\mathbf{e}_k^T.$$

**算法 3：** 用于 RD 计算的 Lanczos 迭代

**输入：** $\mathcal{G}$ $s, t, k$

$\mathbf{v}_0 = 0, \mathbf{v}_1 = \left(\frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}}\right) / \left(\frac{1}{d_s} + \frac{1}{d_t}\right), \beta_1 = 0;$

**对于** $i = 1, 2, \ldots, k$ **do**

  $\mathbf{v}_{i+1} = \mathcal{A}\mathbf{v}_i - \beta_i \mathbf{v}_{i-1}$;

  $\alpha_i = \mathbf{v}_{i+1}, \mathbf{v}_i$;

  $\mathbf{v}_{i+1} = \mathbf{v}_{i+1} - \alpha_i \mathbf{v}_i$;

  $\beta_{i+1} = \|\mathbf{v}_{i+1}\|_2$;

  $\mathbf{v}_{i+1} = \mathbf{v}_{i+1} / \beta_{i+1}$;

**结束**

$$\mathbf{T} = \begin{bmatrix} \alpha_1 & \beta_2 & & 0 \\ \beta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_k \\ 0 & & \beta_k & \alpha_k \end{bmatrix}, \mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_k];$$

**Output:** $\hat{r}_{\mathcal{G}}(s,t) = \left(\frac{1}{d_s} + \frac{1}{d_t}\right) \mathbf{e}_1^T (\mathbf{I} - \mathbf{T})^{-1} \mathbf{e}_1$ as the approximation of $r_{\mathcal{G}}(s,t)$

此外，$\mathbf{V}$ 的列空间张成克雷洛夫子空间 $\mathcal{K}$ $span_{1,\ 1\ \ldots\ k\ 1} = {}^{T}$ $_{k\ 1,\ =}$ $(\mathbf{v}.\mathcal{A})(\mathbf{v}.\mathcal{A}\mathbf{v}_{\prime\prime}.\mathcal{A}\mathbf{v})$，并且 $\mathbf{TV}.\mathcal{A}\mathbf{V}$。

最后，基于引理 3.2 和命题 3.3，我们证明以下定理，该定理给出了 Lanczos 迭代（算法 3）的误差界和时间复杂度。

定理 3.4.（误差保证）设 $\hat{r}_{\mathcal{G}}(s,t)$ 为 *Lanczos* 迭代（算法 *3*）的输出，则误差满足

$$|\hat{r}_{\mathcal{G}}(s,t) - r_{\mathcal{G}}(s,t)| \le \epsilon.$$

当迭代次数设为 $k = O(\sqrt{\kappa}\log \frac{\kappa}{\epsilon})$ 时。此外，当 $k^{1.37} \le m$ 时，算法 *3* 的时间复杂度为 $O(km + k^{2.37}) = O(\sqrt{\kappa}m\log \frac{\kappa}{\epsilon})$。

因此，根据定理 3.4，我们证明了近似值 $\hat{r}_{\mathcal{G}}(s,t)$ 在经过 $k = \sqrt{\kappa}\log \frac{\kappa}{\epsilon}$ 次 Lanczos 递推后满足 $\epsilon$-绝对误差保证。回顾一下，幂法的迭代次数需要达到 $l = \kappa \log \frac{\kappa}{\epsilon}$，因此算法 3 的速度比幂法快 $\sqrt{\kappa}$ 倍。

# 4 一种新颖的 Lanczos Push 方法

## 4.1 Lanczos Push 算法

接下来，我们介绍一种新颖的推送式局部算法，称为 Lanczos Push，它基于一种新近开发的“修剪”技术。在介绍我们的技术之前，我们先解释一下该方法的总体思路。回想一下，Lanczos 迭代会搜索一系列正交向量$\mathbf{v}_1, \ldots, \mathbf{v}_k$，以加速电阻距离计算（即仅依赖于条件数 $\kappa$ 的 $\sqrt{\kappa}$，相比幂法提升了 $\sqrt{\kappa}$ 倍）。现在，为了设计一种电阻距离计算的局部算法，同时保持 Lanczos 迭代的快速收敛速度，我们希望搜索一系列**稀疏**向量 $\hat{\mathbf{v}}_1, \ldots, \hat{\mathbf{v}}_k$，使其与$\mathbf{v}_1, \ldots, \mathbf{v}_k$相似（尽管它们不再必然正交）。图 1 展示了这一思路。

为明确说明我们的方法，让我们仔细考察 Lanczos 迭代（算法 3）的时间复杂度。请注意，在算法 3 的第 3 行，我们执行矩阵-向量乘法 $\mathbf{A}\mathbf{v}_i$，这需要 $O(m)$ 次运算。在算法 3 的第 4 至第 7 行，我们调用内积

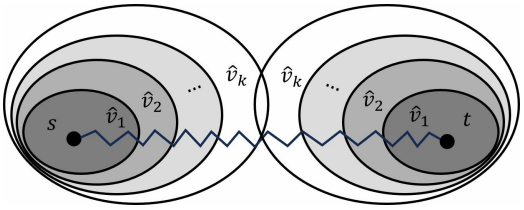



**图 1：我们使用 Lanczos Push 算法计算 $r_{\mathcal{G}}(s,t)$ 的总体思路；$\hat{\mathbf{v}}_1, \ldots, \hat{\mathbf{v}}_k$为稀疏向量。**

以及对$\mathbf{v}_i$进行加法/减法运算，这需要 $O(n)$ 次运算。因此，要将算法 3 转换为局部算法，主要有两项挑战：(i) 在每次迭代中“局部地”实现矩阵-向量乘法 $\mathbf{A}\mathbf{v}_i$。也就是说，我们应仅通过调用图的一小部分来近似 $\mathbf{A}\mathbf{v}_i$。(ii) 在每次迭代中保持$\mathbf{v}_i$ 为稀疏向量。为应对这些挑战，我们提出 Lanczos Push 算法，以局部方式近似这些向量。为此，我们首先定义$\mathbf{v}_i$ 和 $\mathbf{A}\mathbf{v}_i$的稀疏近似。

定义 *4.1*。我们将 $AMV(\mathbf{A}, \hat{\mathbf{v}}_i) \in \mathbb{R}^n$ 定义为近似于 $\mathbf{A}\hat{\mathbf{v}}_i$ 的向量。具体而言，对于每个节点 $v \in \mathbf{V}$，我们定义$AMV(\mathbf{A}, \hat{\mathbf{v}}_i)(v) = \sum_{u \in \mathcal{N}(v)} \hat{\mathbf{v}}_i(u)\sqrt{d_u d_v}\, \mathbb{I}_{|\hat{\mathbf{v}}_i(u)| > \epsilon\sqrt{d_u d_v}}$ 其中，我们将$\mathbb{I}_{|\hat{\mathbf{v}}_i(u)| > \epsilon\sqrt{d_u d_v}}$表示为指示函数，当且仅当$|\hat{\mathbf{v}}_i(u)| > \epsilon\sqrt{d_u d_v}$ 时取值为 1。

定义 *4.2*。我们将 $\hat{\mathbf{v}}_i|_{S_i} \in \mathbb{R}^n$定义为在子集 $S_i \subset \mathbf{V}$ 上受约束的向量 $\hat{\mathbf{v}}_i$。具体而言，对于 $u \in S_i$，$\hat{\mathbf{v}}_i|_{S_i}(u) = \hat{\mathbf{v}}_i(u)$；其他情况下，$\hat{\mathbf{v}}_i|_{S_i}(u) = 0$。其中，$S_i = \{u \in \mathbf{V} : |\hat{\mathbf{v}}_i(u)| > \epsilon d_u\}$。

基于定义 4.1 和 4.2，我们提出以下子集 Lanczos 递推：

$$\hat{\beta}_{i+1}\hat{\mathbf{v}}_{i+1} = AMV(\mathcal{A}, \hat{\mathbf{v}}_i) - \hat{\alpha}_i \hat{\mathbf{v}}_i|_{S_i} - \hat{\beta}_i \hat{\mathbf{v}}_{i-1}|_{S_{i-1}}. \quad (5)$$

在式 (5) 中，根据定义 4.1，$AMV(\mathbf{A}, \hat{\mathbf{v}}_i)$ 是 $\mathbf{A}\hat{\mathbf{v}}_i$ 的稀疏近似；根据定义 4.2，$\hat{\mathbf{v}}_i|_{S_i}$ 是 $\hat{\mathbf{v}}_i$ 的稀疏近似；$\hat{\alpha}_i$ 和 $\hat{\beta}_{i+1}$ 是按照与算法 3 第 4 行和第 6 行相同的方式计算出的标量。综合这些内容，我们给出算法 4 的伪代码。算法 4 的思路很简单：为了在图上局部地实现 Lanczos 迭代，我们执行子集 Lanczos 迭代（式 (5)）。在每次迭代中，我们执行以下两项关键运算：(i) 对矩阵-向量乘法 $\mathbf{A}\hat{\mathbf{v}}_i$ 的近似，我们只搜索使 $\hat{\mathbf{v}}_i(u)$ 非零的节点 $u$。接下来，对于邻居节点 $v \in \mathbf{N}(u)$，我们只累加满足 $\hat{\mathbf{v}}(\ )$ 的边 $(u\ v \in |_i u| >, )$ $\epsilon\sqrt{d_u d_v}$ 4 8 4 $AMV$ , $_i$ E 上的分数，见算法 的第 - 行。我们将 $(\mathbf{A}\hat{\mathbf{v}})$ 表示为经过此运算后 $\mathbf{A}\hat{\mathbf{v}}_i$ 的近似。(ii) 我们选取重要节点的一个子集 $S_i = \{u \in \mathbf{V} : |\hat{\mathbf{v}}_i(u)| > \epsilon d_u\}$，并且仅在子集 $S_i$ 上执行加法/减法约束，见算法 4 的第 9–11 行和第 13–15 行。这些运算保持了计算的稀疏性。

**一个运行示例**。为说明所提出的 Lanczos Push 算法，我们在此给出一个运行示例。考虑一个玩具图，其顶点集 $\mathbf{V} = \{1, 2, 3, 4\}$ 和边集 $\mathbf{E} = \{(1, 2), (1, 3), (1, 4), (2, 3)\}$，如图 2 所示。我们重点计算索引为 1 和 4 的节点之间的电阻距离（RD），根据电阻的定义，其精确值已知为 $r(1, 4) = 1$。为便于演示，我们将算法参数设为 $k =$

**算法 4：** 用于电阻距离计算的 Lanczos Push

**输入：** $\mathcal{G}$, $s$, $t$, $k$, $\epsilon$

$\hat{\mathbf{v}}_0 = 0$, $\hat{\mathbf{v}}_1 = \left(\frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}}\right) / \sqrt{\frac{1}{d_s} + \frac{1}{d_t}}$, $\hat{\beta}_1 = 0$;

**对于** $i = 1, 2, \ldots, k$ **do**

  $S_i = \{u \in \mathrm{V} : |\hat{\mathbf{v}}_i(u)| > \epsilon d_u\}$;

  **对于**节点 $u$，满足 $\hat{\mathbf{v}}_i(u) \neq 0$ **执行**

    **对于** $v \in \mathrm{N}(u)$ 满足 $|\hat{\mathbf{v}}_i(u)| > \epsilon\sqrt{d_u d_v}$ **执行**

      $\hat{\mathbf{v}}_{i+1}(v) = \hat{\mathbf{v}}_{i+1}(v) + \frac{1}{\sqrt{d_u d_v}}\hat{\mathbf{v}}_i(u)$;

    **end**

  **end**

  **对于** $u \in S_{i-1}$ **执行**

    $\hat{\mathbf{v}}_{i+1}(u) = \hat{\mathbf{v}}_{i+1}(u) - \hat{\beta}_i\hat{\mathbf{v}}_{i-1}(u)$;

  **end**

  $\hat{\alpha}_i = \langle \hat{\mathbf{v}}_{i+1}, \hat{\mathbf{v}}_i \rangle$;

  **对于** $u \in S_i$，**执行**

    $\hat{\mathbf{v}}_{i+1}(u) = \hat{\mathbf{v}}_{i+1}(u) - \hat{\alpha}_i\hat{\mathbf{v}}_i(u)$;

  **end**

  $\hat{\beta}_{i+1} = \|\hat{\mathbf{v}}_{i+1}\|_2$;

  $\hat{\mathbf{v}}_{i+1} = \hat{\mathbf{v}}_{i+1}/\hat{\beta}_{i+1}$;

**结束**

$$\hat{\mathbf{T}} = \begin{bmatrix} \hat{\alpha}_1 & \hat{\beta}_2 & & 0 \\ \hat{\beta}_2 & \hat{\alpha}_2 & \ddots & \\ & \ddots & \ddots & \hat{\beta}_k \\ 0 & & \hat{\beta}_k & \hat{\alpha}_k \end{bmatrix}, \hat{\mathbf{V}} = [\hat{\mathbf{v}}_1, \ldots, \hat{\mathbf{v}}_k];$$

**Output:** $r'_{\mathcal{G}}(s,t) = \left(\frac{1}{d_s} + \frac{1}{d_t}\right)\hat{\mathbf{v}}_1^T\hat{\mathbf{V}}(\mathbf{I} - \hat{\mathbf{T}})^{-1}\mathbf{e}_1$ as the approximation of $r_{\mathcal{G}}(s,t)$

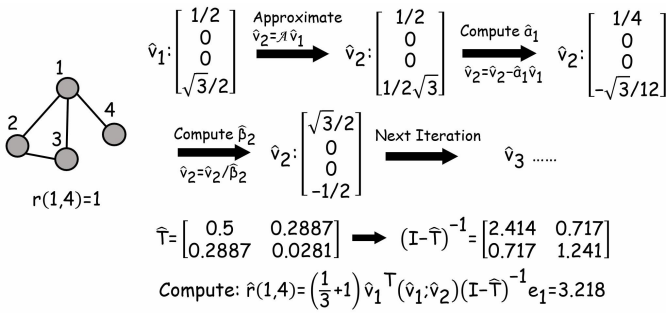



**图 2：我们的 Lanczos Push 算法的运行示例**

2，并将 $\epsilon = 0.25$ 用于 Lanczos Push 算法。首先，我们初始化 $\hat{\mathbf{v}}_1 = \left(\frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}}\right) / \sqrt{\frac{1}{d_s} + \frac{1}{d_t}} = [1/2, 0, 0, \sqrt{3}/2]^T$，并根据算法 4 的第 1 行设置 $\hat{\beta}_1 = 0$。根据第 3 行，我们设置 $S_1 = \{1, 4\}$。接下来，调用第 4–8 行来近似计算矩阵-向量乘积 $\mathbf{A}\hat{\mathbf{v}}_1$。值得注意的是，由于对于 $(u, v) = (1, 2)$ 和 $(1, 3)$，$\lceil \hat{\mathbf{v}}_1(u)/\sqrt{d_u d_v} \rceil = 0.5/\sqrt{6} < 0.25 = \epsilon$，因此不会沿边 $(1, 2)$ 和 $(1, 3)$ 进行计算。完成第 4–8 行的计算后，我们得到 $\hat{\mathbf{v}}_2 = [1/2, 0, 0, 1/2\sqrt{3}]^T$。然后，由于我们在初始步骤中设置了 $\hat{\beta}_1 = 0$，因此跳过第 9–11 行的计算。接下来，根据第 12 行计算 $\hat{\alpha}_1 = \langle \hat{\mathbf{v}}_2, \hat{\mathbf{v}}_1 \rangle = 1/2$。根据第 13–15 行，我们更新 $\hat{\mathbf{v}}_2 = \hat{\mathbf{v}}_2 - \hat{\alpha}_1\hat{\mathbf{v}}_1|_{S_1} = [1/4, 0, 0, -\sqrt{3}/12]^T$。根据第 16 行计算 $\hat{\beta}_2 = \|\hat{\mathbf{v}}_2\|_2 = 1/2\sqrt{3}$，最后根据第 17 行更新 $\hat{\mathbf{v}}_2 = \hat{\mathbf{v}}_2/\beta_2 = [\sqrt{3}/2, 0, 0, -1/2]^T$。随后，我们转向下一次迭代。对于迭代 $i = 2$，我们可以类似地计算 $\hat{\alpha}_2 = 0.0281$；这里省略具体细节，因为所有例程都保持不变。最后，我们通过 $\hat{r}(1,4) = \left(\frac{1}{3} + 1\right)\hat{\mathbf{v}}_1^T[\hat{\mathbf{v}}_1; \hat{\mathbf{v}}_2](\mathbf{I} - \hat{\mathbf{T}})^{-1}\mathbf{e}_1 = 3.218$ 计算近似电阻距离值。请注意，这个结果看起来不准确，主要是因为为了更好地说明问题，我们设置了较大的阈值 $\epsilon = 0.25$。在真实世界图中，当阈值 $\epsilon$ 设置得较小时，我们的 Lanczos Push 算法能够成功降低计算成本，同时不会产生过多的额外误差。理论保证将在下一小节中给出。

## 4.2 Lanczos Push 算法的分析

这里，我们证明，与 Lanczos 迭代相比，我们的 Lanczos Push 算法（算法 4）不会引入显著误差，同时还能大幅缩短运行时间。然而，由于证明较为复杂，这里仅提供概要，详细内容见完整版 [52]。总体而言，理论分析分为两个步骤：(i) 算法 4 的误差界；(ii) 算法 4 的局部时间复杂度。为简便起见，我们记 $C_1 = \max_{u=s,\ t;i\leq k} \|T_i(\mathbf{P})\mathbf{e}_u\|_1$，其中 $T_i(x)$ 表示第一类切比雪夫多项式（定义见附录中的公式 13）。我们记 $C_2 = \max_{i\leq k}(\|\hat{\mathbf{v}}_i\|_1 + \|\mathbf{A}\hat{\mathbf{v}}_i^+\|_1 + \|\mathbf{A}\hat{\mathbf{v}}_i^-\|_1)$，其中 $\hat{\mathbf{v}}_i^+$，$\hat{\mathbf{v}}_i^-$ 分别表示 $\hat{\mathbf{v}}_i$ 的正部和负部。

定理 4.3。 给定 $\epsilon' \in (0,1)$，设置迭代次数 $k = \sqrt{\kappa}\log\frac{\kappa}{\epsilon'}$、参数 $\epsilon = \tilde{\Omega}(\epsilon'/\kappa^{2.25}C_1)$，算法 4 输出的近似值 $r'_{\mathcal{G}}(s,t)$ 与准确电阻距离值 $r_{\mathcal{G}}(s,t)$ 之间的误差可界定为：

$$|r_{\mathcal{G}}(s,t) - r'_{\mathcal{G}}(s,t)| \leq \epsilon'.$$

定理 4.4。 算法 4 执行的总运算次数可界定为 $O(\frac{1}{\epsilon}\sum_{i\leq k}(\|\hat{\mathbf{v}}_i\|_1 + \|\mathcal{A}\hat{\mathbf{v}}_i^+\|_1 + \|\mathcal{A}\hat{\mathbf{v}}_i^-\|_1) + k^{2.37})$。当设置 $k = \sqrt{\kappa}\log\frac{\kappa}{\epsilon'}$、$\epsilon = \tilde{\Omega}(\epsilon'/\kappa^{2.25}C_1)$ 时，该值为 $\tilde{O}(\kappa^{2.75}C_1C_2/\epsilon')$。

上述对 Lanczos Push 算法的理论分析基于一个简单的假设。

假设 1。 我们假设 $\lambda(\hat{\mathbf{T}}) \subset [\lambda_n(\mathbf{A})\ \lambda_2(\mathbf{A})]$。

我们注意到，假设 1 的要求较为宽松。事实上，根据引理 3.2 的证明，对于精确的 Lanczos 迭代（即算法 3；当设置 $\epsilon = 0$ 时，也就是 Lanczos Push 算法），我们总有 $\lambda(\mathbf{T}) \subset [\lambda_n(\mathbf{A})\ \lambda_2(\mathbf{A})]$。我们还在实验中证明，该假设在多个数据集上均成立（详情见第 6 节）。

**理解理论结果。** 乍看之下，由于 $C_1$ 和 $C_2$ 的存在，定理 4.3 和定理 4.4 的结果可能不够清晰，因此我们在此作进一步讨论。首先，我们通过证明 $C_1$ 和 $C_2$ 的上界，给出定理 4.4 的一个更简单版本。

推论 4.5。 算法 4 计算 $r_{\mathcal{G}}(s,t)$ 的 $\epsilon$-绝对误差近似的运行时间为 $\tilde{O}(\kappa^{2.75}\sqrt{nm}/(\epsilon\sqrt{d}))$。具体而言，$C_1 \leq \sqrt{m}$ 和 $C_2 \leq 3\sqrt{n}$。

推论 4.5 表明，Lanczos Push 的运行时间对图规模呈次线性依赖 $m$，同时对条件数呈次三次依赖 $\kappa$。这一结果印证了我们的核心直觉：该算法在以 $m$ 衡量时以次线性时间运行，同时保留了依赖于 $\kappa$ 的更快收敛速度。此外，

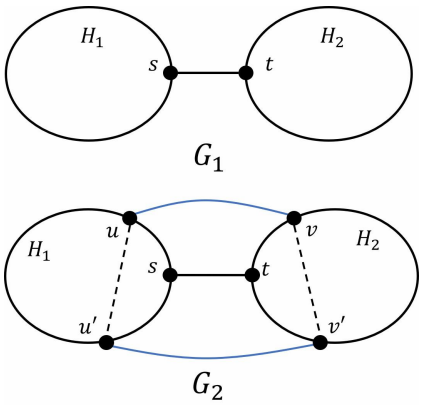



**图 3：下界的构造**

我们的分析首次为以局部方式实现 Lanczos 提供了理论依据。然而，我们的实验结果始终优于推论 4.5 中的最坏情况界，这表明这些理论保证可能并不紧。具体而言，$C_1$ 和 $C_2$ 的实证表现表明，其扩展性优于我们证明中仅使用柯西-施瓦茨不等式所得到的结果。此外，我们的实验（实验 9、第 6 节）揭示了一种有趣的规律：在扩展性较差的图（例如道路网络）上，$C_1$ 和 $C_2$ 往往增长较慢；而在扩展性较好的图（例如社交网络）上，则增长较快。这一发现进一步表明，Lanczos Push 尤其适用于扩展性较差的图。我们希望未来的研究能提出更紧的理论界，或进一步发现能够解释这一现象的结果。

## 5 下界

在本节中，我们建立 $r_{\mathcal{G}}(s,t)$ 的 $\kappa$ 依赖性近似的下界。我们的证明基于 Cai 等人提出的前一项结果 [6]。下面，我们简要介绍在 [6] 中建立的结果。首先，我们考虑两个 $d$ 正则扩展图 $H_1, H_2$（即，$H_1, H_2$ 的条件数为 $\kappa = O(1)$）。具体而言，我们选择 $H_1$，$H_2$，使其成为定义在 $n/2$ 个顶点上的 Ramanujan 图 [28] 的两个副本（三正则图，具有 $\kappa = O(1)$）。接下来，我们通过一条边 $(s,t)$ 连接节点 $s \in H_1$ 和 $t \in H_2$，生成图 $G_1$。显然，$r_{G1}(s,t) = 1$。为了构造 $G_2$，我们随机选择一条边 $(u,u') \in H_1$ 和 $(v,v') \in H_2$。我们移除边 $(u,u')$ 和 $(v,v')$，并添加两条新边 $(u,v)$ 和 $(u',v')$。然后，[6] 证明，经过此操作后，所得图 $G_2$ 的有效电阻 $r_{G_2}(s,t) \le 0.99$。然而，任何能够以正概率区分 $G_1$ 和 $G_2$ 的算法都至少需要 $\Omega(n)$ 次查询。为便于理解，我们将这一结果总结为以下定理。

定理 5.1.（*[6]*中的定理 *1.1*）如果$\mathcal{G}_1$和$\mathcal{G}_2$按上述方式构造，那么我们有 $|r_{\mathcal{G}_1}(s,t) - r_{\mathcal{G}_2}(s,t)| \ge 0.01$。但是，任何能够以概率 $\ge 0.6$区分$\mathcal{G}_1$ 和$\mathcal{G}_2$ 的（随机化）算法，至少需要 $\Omega(n)$ 次查询。

根据定理 5.1，我们可以证明 $G_1$ 的条件数为 $\kappa = \Theta(n)$，如下引理所述。

引理 5.2。 如果按照上述方式构造$\mathcal{G}_1$，则有 $\kappa(\mathcal{G}_1) = \Theta(n)$。

**表 2：数据集统计**

| 数据集 | $n$ | $m$ | $m/n$ | 条件数 $\kappa$ |
|---|---|---|---|---|
| Dblp | 317,080 | 1,049,866 | 3.31 | 717.62 |
| YouTube | 1,134,890 | 2,987,624 | 2.63 | 907.85 |
| LiveJournal | 4,846,609 | 42,851,237 | 8.84 | 2785.51 |
| Orkut | 3,072,626 | 11,718,5083 | 38.13 | 381.46 |
| 电网 | 4,941 | 6,594 | 1.33 | 7299.27 |
| RoadNet-CA | 1,965,206 | 2,766,607 | 1.41 | 64516.12 |
| RoadNet-PA | 1,088,092 | 1,541,898 | 1.42 | 24390.24 |
| RoadNet-TX | 1,379,917 | 1,921,660 | 1.39 | 28985.5 |
| ER | $10^2 \sim 10^6$ | $n$ 对数 $n$ | 对数 $n$ | $O(1)$ |
| BA | $10^2 \sim 10^6$ | $n$ 对数 $n$ | 对数 $n$ | $O(1)$ |

结合定理 5.1 和引理 5.2，我们可以证明，对于任意 $\epsilon < 0.01$，任何算法计算 $\epsilon$-近似电阻距离值 $r_{\mathcal{G}}(s,\ t)$ 至少需要 $\Omega(\kappa)$ 次查询。

定理 5.3。 任何以绝对误差 $\epsilon \le 0.01$ 近似 $r_{\mathcal{G}}(s,t)$ 且成功概率为 $\ge 0.6$ 的算法，都需要 $\Omega(\kappa)$ 次查询。

这表明，局部算法的时间复杂度确实与条件数 $\kappa$相关，而以往关于局部 RD 计算的研究并未强调这一点。我们预计，这个 $\Omega(\kappa)$ 并不紧。遗憾的是，据我们所知，目前还没有更好的构造，因此，如何使上界和下界相匹配仍是一个值得探索的未来方向。

## 6 实验评估

### 6.1 实验设置

我们使用了 8 个公开可用、规模各异的数据集[4]（表 2），其中包括 4 个社交网络（Dblp、YouTube、LiveJournal、Orkut），它们是各种图算法[20, 43, 44, 48, 49] 的标准基准测试；另外 4 个数据集则代表了对以往 RD 算法而言较难处理的案例（由于规模较大$\kappa$）[21]。其中，RoadNet-CA、RoadNet-PA 和 RoadNet-TX 是道路网络，powergrid 是一种基础设施网络，其统计特征与道路网络相似。此外，我们还人工生成了经典的埃尔德什-雷尼（ER）图和巴拉巴西-阿尔伯特（BA）图，其规模为 $n$，变化范围从 $10^2 \sim 10^6$ 到 $m = n\log n$，用于可扩展性测试。为计算这些数据集的条件数，我们使用标准幂法计算归一化邻接矩阵的第二大特征值 $\mu_2$；根据定义，条件数为 $\kappa = \frac{2}{1-\mu_2}$。我们将幂法的阈值设为 $10^{-9}$，以确保对 $\mu_2$ 的近似足够准确。为评估不同算法的近似误差，我们使用幂法（PM）并采用足够大的截断步数 $N = 20000$，计算真实值 RD 值。我们将提出的算法 Lanczos 迭代（Lz）和 Lanczos Push（LzPush）与 6 种最先进水平（SOTA）的基线方法进行比较：PM [49], GEER [49], Push [20], BiPush [20],，以及两项近期研究提出的 BiSPER [13] 和 FastRD [24]。由于 [20, 49] 中提出的算法在所有数据集上的表现均优于 [33]中提出的算法，因此我们不再将所提算法与 [33]中提出的算法进行比较。我们随机生成 50 个源节点

[4]所有 8 个数据集均可从 http://snap.stanford.edu/ 和 http://konect.cc/networks/ 下载。

和 50 个汇节点作为查询集，并报告不同算法在这些节点上的平均性能。根据定义 2.2，我们使用绝对误差（记作“绝对误差”）评估不同算法的估算误差。所有实验均在一台运行 Linux 20.04 的服务器上进行，该服务器配备英特尔 2.0 GHz 中央处理器和 128GB 内存。所有算法均使用 C++ 实现，并通过 GCC 9.3.0 编译，采用 -O3 优化。源代码可在 https://github.com/Ychun-yang/LanczosPush 获取。

## 6.2 总体实证结果

**实验 1：不同算法在无权图上的查询时间**。在本实验中，我们将所提出的算法 Lz 和 LzPush 与先前研究 [13, 20, 24, 49]提出的 6 种最先进算法进行比较：PM、GEER、Push、BiPush、BiSPER 和 FastRD。对于 PM、GEER、Push 和 BiPush，我们遵循 [20, 49]，将参数 $\epsilon$ 从 $10^{-1}$ 变化到 $10^{-5}$。对于 BiSPER 和 FastRD，我们遵循 [13, 24]，将 $\epsilon$ 从 0.9 变化到 $10^{-3}$。对于 FastRD，由于在高精度计算下，它在百万规模图上的存储开销难以接受，因此对于大型数据集，我们仅在较大的 $\epsilon$ 下比较 FastRD，例如 $\epsilon = 0.9, 0.5, 0.1$，并遵循 [24]中的设置。对于条件数较小的数据集（即四个社交网络），为公平比较，我们对所提算法的参数设置如下：对于 Lz，我们改变迭代步数 $k = [5, 10, 15, 20, 30]$；对于 LzPush，我们改变 $\epsilon = [0.1, 0.05, 0.01, 0.005, 0.001]$ 并固定 $k = 20$。图 4 报告了不同算法在绝对误差指标下计算电阻距离的查询时间。我们得到以下观察结果：(i) 所提出的算法 LzPush 效率最高，在所有数据集上的速度均比其他竞争算法快 5× 至 10× 倍。(ii) 由于 LzPush 的时间复杂度与顶点和边的数量无关，因此对于规模各异的数据集，我们观察到 LzPush 的性能几乎保持不变。这与我们的理论分析一致（LzPush 是一种局部算法）。另一方面，Lz 和 PM 的性能取决于 $m$，随着数据集规模增大（例如 LiveJournal、Orkut），其查询时间也会增加。

对于条件数较大的数据集（即三个道路网络和电网），为确保比较公平，我们设置参数如下：对于 Lz，我们调整迭代步数 $k = [20, 40, 60, 80, 100]$；对于 LzPush，我们调整 $\epsilon = [10^{-3}, 5 * 10^{-4}, 10^{-4}, 5 * 10^{-5}, 10^{-5}]$ 并固定 $k = 100$。图 5 报告了不同算法在绝对误差指标下进行电阻距离计算的查询时间。我们有如下观察：(i) 我们提出的算法 LzPush 仍然效率最高，而所有先前的局部算法（GEER、Push、BiPush、BiSPER）在三个道路网络上都无法快速给出高质量结果（例如，绝对误差 $< 10^{-1}$）。原因如下：由于道路网络的条件数 $\kappa$ 较大，基于随机游走采样的方法需要设置非常大的截断步数 $l$ 和非常多的采样次数 $n_r = \tilde{O}(l^2/\epsilon^2)$。这使得 GEER 效率低下。尽管双向技术提高了随机游走的效率，BiSPER 也有类似的表现。对于 Push 和 BiPush，这些算法是启发式算法，其时间复杂度与命中时间 $h(s, v) + h(t, v)$ 有关，其中 $v$ 是预先选定的地标节点。然而，在道路网络上，$h(s, v)$ 可能为 $O(n)$，这使得 Push 和 BiPush 的效率非常低，尤其是在高精度计算时。尤为值得注意的是，在三个道路网络上，我们的 LzPush 算法快 1000× 倍于幂法，且比 Lz 快 50× 倍。(ii) LzPush 在四个数据集上的性能相近，而 Lz 和幂法的性能则与 $m$ 有关。具体而言，Lz 和幂法在较小的电网图上速度更快，但在三个较大的道路网络上速度较慢。这与社交网络上的结果一致。

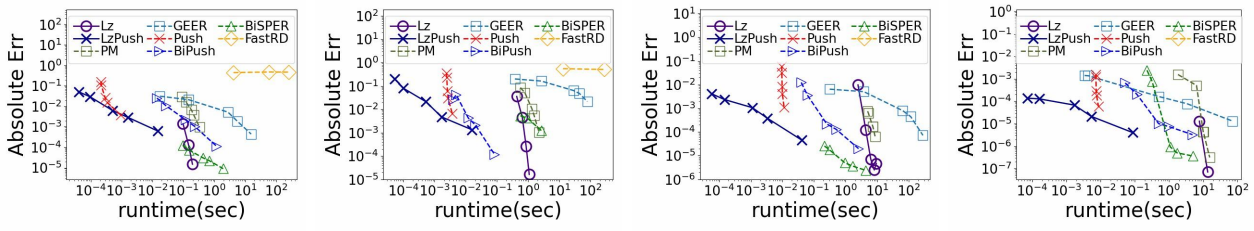
**实验 2：加权图上不同算法的查询时间**。在本实验中，我们在加权图上评估所提出的算法，并与现有基线方法进行比较。遵循以往工作的常见做法 [45], ，我们通过为每条边 $e \in$ E赋予权重来生成无权图的加权版本，该权重等于包含 $e$[5]的三角形数量。图 6 展示了 Dblp 和 LiveJournal 数据集上的结果。在其他数据集上也可以观察到类似结果。如图 6 所示，结果与无权图实验的表现一致：在所有比较的方法中，我们的 LzPush 算法仍然是效率最高的算法。

**实验 3：合成图上的可扩展性测试**。在本实验中，我们比较所提出的 Lz 和 LzPush 算法的可扩展性。我们使用经典的 ER 图和 BA 图，并改变 $n$，范围从 $10^2$ 到 $10^6$ 以及 $m = n \log n$。我们设置迭代次数$k = 20$和阈值 $\epsilon = 0.005$，并比较 Lz 与 LzPush 的运行时间。图 7 展示了结果。可以看出，Lz 的运行时间相对于图规模 $n$几乎呈线性增长，而 LzPush 的运行时间相对于 $n$的增长呈次线性。这些结果进一步表明，Lz 是近线性的全局算法，而 LzPush 是局部算法，这与我们的理论分析一致。因此，这些发现有力地证明，所提出的 Lz 和 LzPush 在超大规模图上具有出色的可扩展性。

**实验 4：不同数据集上的内存使用测试**。在本实验中，我们评估了所提出的算法 Lz 和结合 PM 的 LzPush 在低精度情形（图 8 (a)）和高精度情形（图 8 (b)）下的内存使用量。具体而言，对于低精度情形，我们将 $k = 5, \epsilon = 10^{-1}$ 用于社交网络，并将$k = 20, \epsilon = 10^{-3}$ 用于道路网络；对于高精度情形，我们将 $k = 30, \epsilon = 10^{-3}$ 用于社交网络，并将 $k = 100, \epsilon = 10^{-5}$ 用于道路网络。从图 8 中，我们得到以下观察结果：(1) Lz 和 LzPush 的内存依赖关系均随图规模呈线性增长，与 PM 的 $3n$ 需求（存储三个向量）相当，其中 LzPush 的使用量略高于 Lz；LzPush 的内存使用量略高于 Lz。(2) 在不同精度设置下，内存消耗几乎保持一致，因为我们的 Lz 实现每次迭代仅维护三个向量（$\mathbf{v}_{i-1}$、$\mathbf{v}_i$、$\mathbf{v}_{i+1}$），而不是完整的矩阵 $\mathbf{V} = [\mathbf{v}_0, ..., \mathbf{v}_k]$。LzPush 的额外内存需求主要来自迭代过程中存储候选集 $S_i$ 和 $\mathbf{v}_i$ 的非零项，尽管总体空间复杂度仍为 $O(n)$。因此，即使面对大规模数据集，我们的算法仍能保持较高的内存效率，因为其 $O(n)$存储需求主要由图数据本身所需的 $O(m)$空间决定。

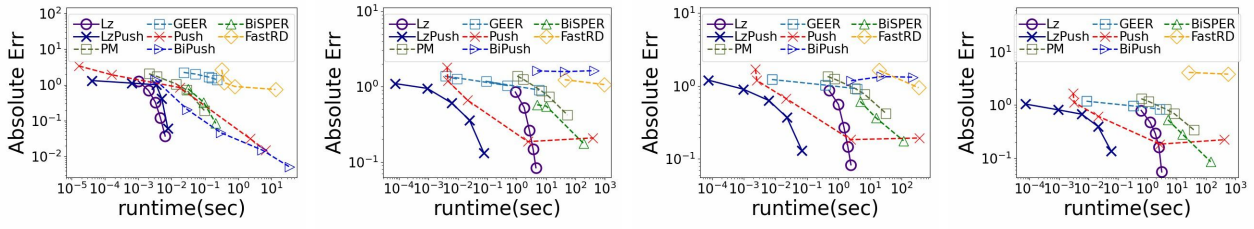
**实验5：高精度下Lz 和LzPush 的稳定性测试**。在本实验中，我们考察所提出的 Lz 和 LzPush 算法的误差稳定性。具体而言，我们将 Lz 和 LzPush 的迭代次数 $k$ 从 5 调整到 150，并设置 $\epsilon_1 = 10^{-3}, \epsilon_2 = 10^{-5}$,

[5]如果 $e$ 不属于任何三角形，我们设置 $w(e) = 1$ 以确保网络连通。



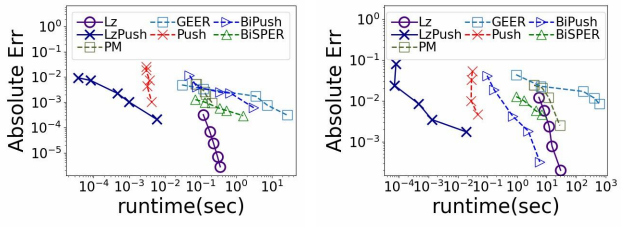

(a) Dblp (b) YouTube (c) LiveJournal (d) Orkut

**图 4：条件数较小的图上不同算法的查询时间 $\kappa$**



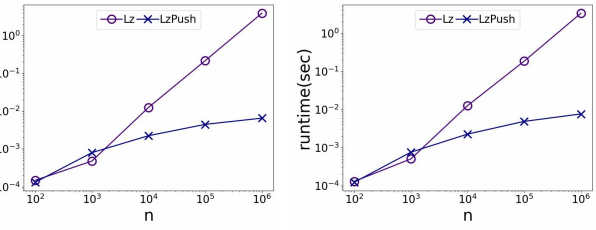

(a) 电网 (b) 道路网络-加利福尼亚州 (c) RoadNet-PA (d) RoadNet-TX

**图 5：条件数较大的图上不同算法的查询时间 $\kappa$**




（a）权重-Dblp （b）权重-LiveJournal

**图6：不同算法在加权图上的查询时间**

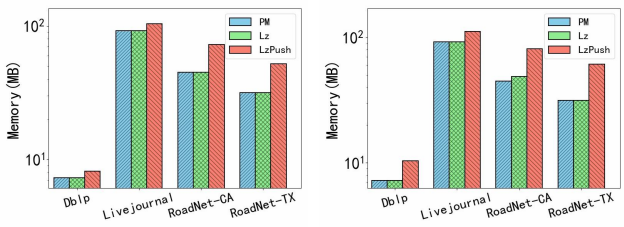



(a) ER (b) BA

**图7：合成图上的可扩展性测试**

$\epsilon_3 = 10^{-7}$ 使 LzPush 达到不同的绝对误差（见图 9）。结果表明，当 $k$ 足够大且$\epsilon$足够小时，两种 Lanczos 算法都能获得高精度解，且收敛速度比幂法快 $\sqrt{\kappa}$ 倍。[6] 因此，Lz 和 LzPush

[6]与先前研究比较，[13, 20, 21, 33, 49], 社会网络的 $10^{-7}$ 绝对误差和 $10^{-3}$ 道路网络的绝对误差已经具有非常高的精度，适用于



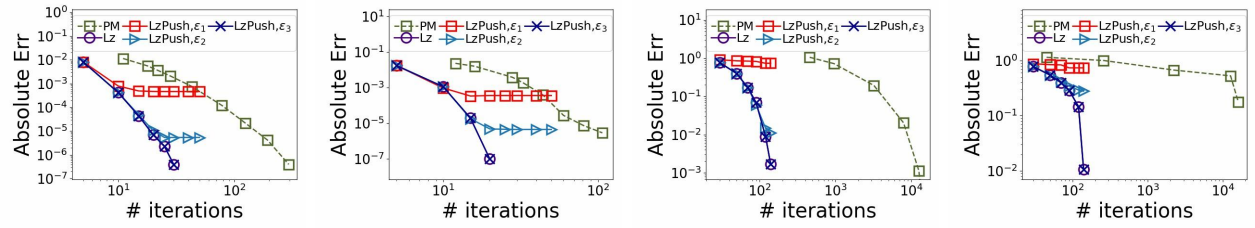

(a) 低精度 （b）高精度

**图8：不同数据集上的内存使用测试**

在高精度计算下都具有稳定性。我们的发现与先前的研究 [27, 29] 一致：Lanczos方法在矩阵函数计算中保持数值稳定。在我们的研究中，RD 计算的稳定性源于这一现象，因为 $\mathbf{L}^{\dagger}(\mathbf{e}_s - \mathbf{e}_t)$ 是矩阵函数。因此，我们的算法在高精度计算中不会出现不稳定问题。此外，我们还观察到，随着 $k$ 增大，Lz 产生的绝对误差呈指数级下降，但在 $\epsilon = 10^{-3}$ 和 $10^{-5}$下，LzPush 产生的绝对误差没有显著变化。这些结果表明，Lz 的效率主要取决于 $k$，而当 $k$较大时，LzPush 产生的误差主要取决于 $\epsilon$，这与我们的理论分析一致。

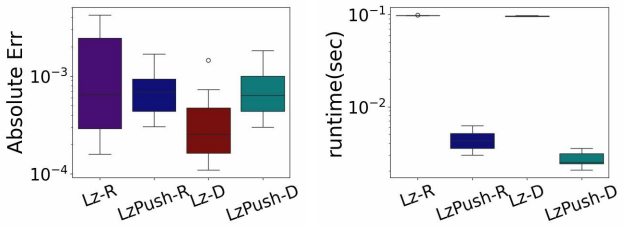
**实验6：在不同密度和节点选择策略下对Lz 和LzPush 进行稳定性测试。本实验评估Lz 和LzPush 在以下方面的稳定性：(i) 不同的节点选择**

电阻距离计算。这是因为 幂法 需要经过超过 $10^3$ 次迭代才能在社交网络上达到真实值计算的这一误差保证，而在道路网络上则需要超过 $10^4$ 次迭代。



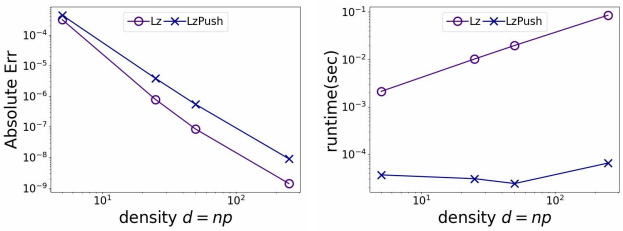

(a) Dblp (b) LiveJournal (c) 电网 (d) RoadNet-CA

**图9：Lz 和 LzPush 的稳定性测试，迭代次数变化时 $k$ 和 $\epsilon$**




(a) Dblp，绝对误差 (b) Dblp，运行时间

**图10：Lz 和LzPush 采用不同查询节点选择策略时的性能。Lz −R 和LzPush −R 分别表示Lz和LzPush 以均匀随机方式选择源节点 $s$ 和汇节点 $t$。Lz − D 和LzPush −D 分别表示Lz 和LzPush选择度最高的源节点 $s$ 和汇节点 $t$。**




(a) ER，绝对误差 (b) ER，运行时间

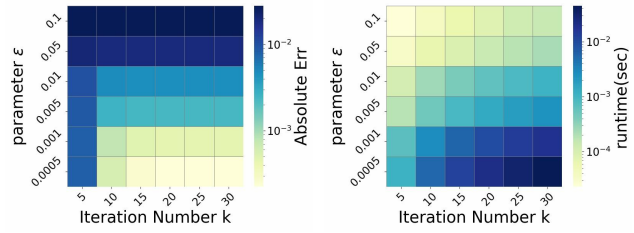
**图11：在密度变化的情况下，Lz 和LzPush 在 $\mathbb{G}(n,\ p)$ 上的性能，其中密度定义为 $d = np$。**

策略；（ii）数据集的密度。具体来说，对于节点选择策略，我们探讨了两种选择源节点$s$和汇节点 $t$的方法：（i）选择度最高的 10 个节点；（ii）从图中随机选择 10 个节点。对于 Lz 和 LzPush，我们设置参数 $\epsilon = 10^{-3}$ 和 $k = 15$。图 10 使用箱线图展示了 Dblp 上查询质量和查询时间的分布。由此可见，我们得到以下观察结果：（i）Lz 和 LzPush 产生的误差对节点选择策略并不敏感。（ii）对于不同的源节点/汇节点选择策略，查询时间和绝对误差没有显著差异。总体而言，Lz 和 LzPush 的性能并不严重依赖于$s$，$t$ 节点选择策略。对于密度测试，我们考虑 ER 随机图 $\mathbb{G}(n,\ p)$，包含 $n = 1000$ 个节点。我们定义密度 $d = np$，并改变 $d = [5, 25, 50, 250]$ 以生成密度不同的数据集。我们为 Lz 和 LzPush 分别设置 $k = 5$ 和 $\epsilon = 0.1$。图 11 展示了结果。我们得到以下观察结果：Lz 的运行时间随图的密度线性增长，而 LzPush 对密度并不敏感。此外，密度 $d$ 增大时，绝对值会减小。这主要是因为稠密图的精确电阻距离值较小。

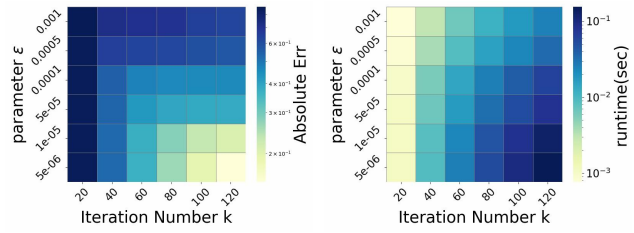
**实验7：LzPush在不同 $k$ 和 $\epsilon$条件下的性能。** 在本实验中，我们考察了LzPush在两个典型数据集Dblp和RoadNet-CA上，针对不同 $k$ 和 $\epsilon$ 设置的性能。由于其他数据集呈现出相似趋势，因此不予考虑。图12通过热力图展示了这一结果。如图所示，我们得到以下观察结果：（1）观察图12（a）和12（c）中的每一行可知，在这两个数据集上，对于每个固定的$\epsilon$，随着 $k$ 增加，LzPush产生的绝对误差并未大幅下降。不过，对于预先选定的特定 $k$（例如，Dblp上的 $k = 20$和RoadNet-CA上的 $k = 120$），$\epsilon$的不同设置确实会影响LzPush的性能。具体而言，$\epsilon$越小，绝对误差越低。（2）LzPush的运行时间（即图12（b）和12（d））均受到 $\epsilon$ 和 $k$的影响。具体而言，较大的$k$和较小的 $\epsilon$ 会导致LzPush的运行时间更长。因此，这些结果与我们的理论分析一致。

**实验8：检验假设 1 的合理性。**在本实验中，我们检验假设 1 是否在数值实验中成立，该假设要求 $\lambda(\hat{\mathbf{T}}) \subset [\lambda_n(\mathrm{A})\ \lambda_2(\mathrm{A})]$。为便于直观展示，我们考察$\mathbf{I} - \mathbf{T}$的特征值，并发现 $\lambda(\mathbf{I} - \hat{\mathbf{T}}) \subset [\lambda_2(\mathrm{L})\ \lambda_n(\mathrm{L})]$。对于社交网络，我们固定$k = 15$；对于道路网络，我们固定 $k = 100$，同时让 $\epsilon$取不同值。图13显示$\mathbf{I} - \hat{\mathbf{T}}$的特征值始终落在 $\lambda(\mathrm{L})$的范围内，并且随着 $\epsilon$减小，极端特征值趋近于 $\lambda_2(\mathrm{L})$和 $\lambda_n(\mathrm{L})$。这些结果支持假设 1 的合理性，并表明 Lanczos Push 算法也可能适用于近似计算极端特征值。

**实验 9：测试 $C_1$ 和 $C_2$的值。** 回顾一下，在理论分析中，LzPush 的运行时间上界为 $\tilde{O}(k^{2.75}C_1C_2/\epsilon)$，其中还涉及另外两个数值 $C_1 = \max_{u=s,\ t;i\le k} \|T_i(\mathbf{P})\mathbf{e}_u\|_1$ 以及 $C_2 = \max_{i\le k}(\|\hat{\mathbf{v}}_i\|_1 + \|\mathbf{A}\hat{\mathbf{v}}_i^+\|_1 + \|\mathbf{A}\hat{\mathbf{v}}_i^-\|_1)$。在本实验中，我们测试这两个数值在不同 $k$和 $\epsilon$下的取值。具体而言，对于 $C_1$，我们将社交网络中的 $k$从 5 调整到 30，将道路网络中的 $k$从 20 调整到 120，结果如图14所示。我们有以下观察：(i) $\|T_i(\mathbf{P})\mathbf{e}_u\|_1$ 会随着图规模和 的增大而增大，但在实验中的扩展表现优于其理论最坏情况推论 4.5。(ii) 在扩张性较差的图（道路网络）中，$C_1$的增长速度低于扩张性较好的数据集（社交网络）。

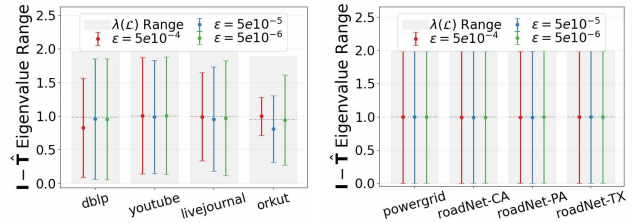



(a) Dblp，绝对误差　(b) Dblp，运行时间




(c) RoadNet-CA，绝对误差　(d) RoadNet-CA，运行时间

**图 12：** LzPush 在不同 $\epsilon$ 和 $k$ 下的性能



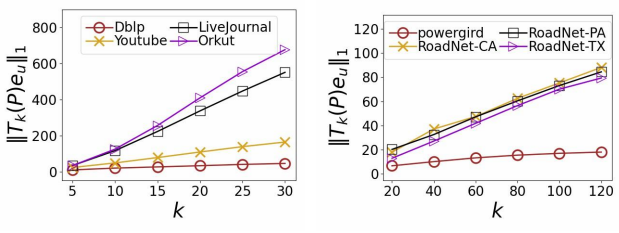

(a) 社交网络　(b) 道路网络

**图 13：测试不同数据集下 $\mathbf{I}-\hat{\mathbf{T}}$ 的特征值范围及其变化情况 $\epsilon$**

对于 $C_2$，我们固定社交网络中的$k = 15$、道路网络中的 $k = 100$，并改变 $\epsilon$，结果如图 15 所示。观察结果与 $C_1$相似：$\|\hat{\mathbf{v}}_i\|_1$，$\|\mathbf{A}\hat{\mathbf{v}}_i\|_1$会随着图规模增大和 $\epsilon$减小而增大，并且其在道路网络上的增长速度较慢。这些观察进一步表明，与之前的算法相比，LzPush 在道路网络上的表现更好。

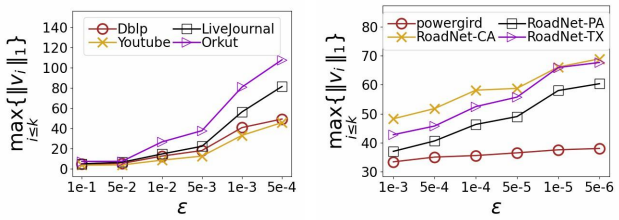
## 6.3 案例研究：道路网络上的鲁棒路由

所提方法的一个有趣应用是道路网络上的鲁棒路由。近期一项研究 [39] 表明，通过电流流生成一组备选路线具有以下优势：(i) 绕行度。每条备选路径的表现都不会比最短路径差太多。(ii) 多样性。备选路径彼此之间有足够的差异。(iii) 鲁棒性。在随机删除边的情况下，以较高概率，备选路径中会有一条可行路线。具体而言，他们的算法将电流

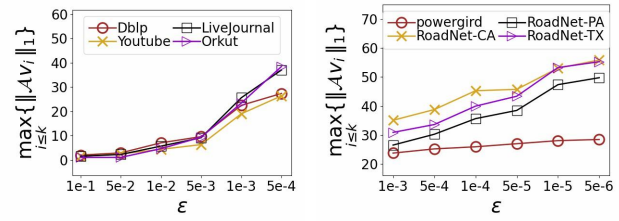



(a) 社交网络　(b) 道路网络

**图 14：测试 $k$ 变化时 $\max_{u=s,\ t;i\le k}\|T_i(\mathbf{P})\mathbf{e}_u\|_1$ 的值**




(a) 社交网络，$\|\hat{\mathbf{v}}_i\|_1$　(b) 道路网络，$\|\hat{\mathbf{v}}_i\|_1$




(c) 社交网络，$\|\mathbf{A}\hat{\mathbf{v}}_i\|_1$　(d) 道路网络，$\|\mathbf{A}\hat{\mathbf{v}}_i\|_1$

**图 15：测试 $\epsilon$ 变化时 $\max_{i\le k}\|\hat{\mathbf{v}}_i\|_1$ 和 $\max_{i\le k}\|\mathcal{A}\hat{\mathbf{v}}_i\|_1$ 的值**

**算法 5：**使用 Lanczos 迭代的鲁棒路由。

**输入：** $\mathcal{G}$ $s$，$t$，$k$，$l$

$\hat{\boldsymbol{\phi}} \leftarrow$ Lanczos 迭代（G $s$，$t$，$k$），最终输出修改为 $\hat{\boldsymbol{\phi}} = \frac{1}{\sqrt{d_s}} + \frac{1}{d_t}\mathbf{D}^{-1/2}\mathbf{V}(\mathbf{I}-\mathbf{T})^{-1}\mathbf{e}_1$;

计算 $\hat{\mathbf{f}}$ by $\hat{\mathbf{f}}(e) = \hat{\boldsymbol{\phi}}(u) - \hat{\boldsymbol{\phi}}(v)$ 对任意 $e=(u,\ v)\in\mathrm{E}$;

**对** $i \in [2l]$ **执行**

　求一条路径 P，使从……出发的最小流量最大化 $\hat{\mathbf{f}}$ 沿着其各条边，并将其从……中移除 $\hat{\mathbf{f}}$;

　如果不存在 P，则中断;

**结束**

**输出：** $l$ 条成本最小的路径

流量计算是一个关键子程序，与我们的 RD计算问题密切相关。

定义*6.1*。给定两个节点 $s$，$t$，势向量 $\boldsymbol{\phi}\in\mathbb{R}^n$定义为 $\boldsymbol{\phi}=\mathbf{L}^{\dagger}(\mathbf{e}_s-\mathbf{e}_t)$。电流向量 $\mathbf{f}\in\mathbb{R}^m$定义为 $\mathbf{f}(e)=\boldsymbol{\phi}(u)-\boldsymbol{\phi}(v)$，对于每条边 $e=(u,\ v)\in\mathrm{E}$。

我们表明，只需调整一行，就可以将所提出的 Lanczos 算法直接推广，以计算 $\boldsymbol{\phi}$（从而计算 $\mathbf{f}$）。

**表 3：不同鲁棒路由方法的比较**

| 指标/方法 | 电流 | Penalty | Plateau |
|---|---|---|---|
| 绕行度 | 1.15 | 1.64 | 1.09 |
| 多样性 | 0.93 | 0.98 | 0.51 |
| 鲁棒性 | 0.95 | 0.85 | 0.52 |
| 运行时间 | 0.039 | 0.036 | 0.099 |

参见算法 5。经过此修改后，我们证明 Lanczos 算法能够计算出 $\mathbf{f}$ [7] 的精确近似。

定理 6.2。 算法 *5* 在迭代次数 $k = O(\sqrt{\kappa}\log\frac{m}{\epsilon})$时，计算出$\hat{\mathbf{f}}$，使得 $\|\mathbf{f} - \hat{\mathbf{f}}\|_1 \leq \epsilon$。

接下来，我们采用与 [39] 相同的策略，利用电流$\mathbf{f}$计算 $l$ 条鲁棒路径：我们迭代地寻找一条路径，使从 $s$ 到 $t$的最小流量最大化，然后从该路径经过的边中移除相应的最小流量值。我们重复此过程 $2l$ 次，最后找出成本最低的 $l$ 条路径。我们提供了一个以从 OpenStreetMap 提取的波士顿数据为例的运行示例 [12]。我们随机选择两个节点索引 $s, t$。我们选择 $l = 10$ 条备选路线，并将 $k = 150$ 用于 Lanczos 算法；根据以下三个指标 [39]，我们比较基于电流的路由与最先进的最短路径路由方法：Penalty 和 Plateau：（i）绕行度：备选路径的平均长度与 $s, t$-最短路径长度之比；（ii）多样性：所有备选路径之间的平均成对 Jaccard 相似度；（iii）鲁棒性：随机删除边后仍存在一条备选路径的概率。具体而言，在我们的实验中，每条边都以0.01的概率独立删除。实验结果见表 6。如表所示，与 Penalty 相比，基于电流的路由输出的路径具有更好的鲁棒性和绕行度；与 Plateau 相比，其路径具有更好的鲁棒性和多样性。此外，使用 Lanczos 算法的运行时间与最先进方法的运行时间相当。我们注意到，在先前的研究中，基于电流的路由的主要瓶颈是电势 $\boldsymbol{\phi}$ [39]的计算。我们的算法克服了这一瓶颈，从而实现了一种快速且实用的备选路由解决方案。

## 7 相关工作

**电阻距离估计及其应用**。关于电阻距离估计的研究由来已久。目前的电阻距离估计算法大致可分为两类：（i）针对多个顶点对的电阻距离计算 [9, 14, 17, 19, 40, 51]；（ii）针对单源或单个顶点对的电阻距离计算 [13, 20, 21, 33, 49]。具体而言，对于第一个问题，针对一般无向图，最先进的理论结果是一个运行时间为 $\tilde{O}(m + (n + |S|)/\epsilon^{1.5})$ 的算法，可近似计算 $\epsilon$ 个电阻距离值对 $S$，误差为 [9]。对于扩展图，在保持相同误差保证的同时，时间复杂度最近已降至 $\tilde{O}(\sqrt{nm}/\epsilon + |S|)$ [51]。对于第二个问题，针对给定的一对节点 $s$，$t$，最先进算法 [33, 49] 可在 $\tilde{O}(1/\epsilon^2)$ 时间内估计 $r_{\mathcal{G}}(s, t)$，并提供 $\epsilon$ 的绝对误差保证；对于扩展图。本文着重提升非扩展图（即，$\kappa$ 不是$O(1)$）上局部算法的效率，因此我们的研究与以往这些研究各自具有独立意义。电阻距离计算广泛用作图稀疏化 [40], 、图聚类 [1], 、随机生成树计数 [9, 19], 、道路网络的鲁棒路由 [15] 以及理解图神经网络中的过度压缩 [4]等任务的子程序。

**Lanczos 迭代**。 Lanczos 迭代理论上和实验上都表现出强大的能力。在数值分析中，理解有限精度下 Lanczos 迭代的稳定性是一项关键问题。例如， [26, 29]分析了有限精度下 Lanczos 迭代在矩阵函数近似中的误差传播； [32]研究了 Lanczos 迭代的特征值近似误差界； [7] 研究了正则谱分布下 Lanczos 算法的稳定性。然而，本文与仅分析 Lanczos 迭代引起的舍入误差不同，我们通过子集 Lanczos 递推（式 5）主动引入误差，从而使算法能够在局部实现。因此，我们提出的技术相较于以往研究具有独立的研究价值。近年来，Lanczos 迭代也应用于各种图算法中。例如， [30] 使用 Lanczos 迭代计算 Heat Kernel PageRank 值； [5] 使用 Lanczos 迭代计算 Katz 分数和往返时间； [41] 使用随机 Lanczos 求积估计冯·诺依曼图熵； [3, 8] 使用随机 Lanczos 求积设计线性时间算法，以近似对称矩阵的谱密度； [38] 则在子图上应用 Lanczos 迭代，计算用于局部聚类的极端特征向量。然而，以上研究都直接将 Lanczos 算法作为子过程应用，而我们提出了首个通过局部实现来修改 Lanczos 算法的算法。最后需要指出，尽管我们的算法主要用于计算电阻距离值，但也可以推广到更多应用场景。这些推广留待未来工作。

## 8 结论

本文提出了两种高效的 RD 计算算法，即 Lanczos 迭代和 Lanczos Push。具体而言，我们首先证明，经典 Lanczos 迭代可用于计算 RD 值。我们证明，Lanczos 迭代可在 $\tilde{O}(\sqrt{\kappa}m)$ 时间内实现，并以 $\epsilon$-绝对误差保证近似 RD 值，其速度比传统幂法快 $\sqrt{\kappa}$ 倍。随后，我们提出一种新颖的子集 Lanczos 递推技术，以实现 RD 计算的局部算法。我们证明，在一些适度的假设下，我们的 Lanczos Push 算法可在 $\tilde{O}(\kappa^{2.75}C_1C_2/\epsilon)$ 时间内实现，并以 $\epsilon$ 绝对误差近似 RD 值，相比最先进方法提升了 $\kappa^{0.25}$ 。大量实验结果表明，我们的算法显著优于现有的 RD 计算最先进方法，尤其适用于大型图以及条件数较大的图 $\kappa$。

[7]为简化起见，我们仅给出 Lanczos 算法的误差分析，不过，所提出的 Lanczos Push 算法也可以修改为计算电流。

# 参考文献


[1] Vedat Levi Alev、Nima Anari、Lap Chi Lau 和 Shayan Oveis Gharan。利用有效电阻进行图聚类。arXiv 预印本 arXiv:1711.06530，2017。[2] Alexandr Andoni、Robert Krauthgamer 和 Yosef Pogrow。关于次线性时间求解线性系统。arXiv 预印本 arXiv:1809.02995，2018。[3] Rajarshi Bhattacharjee、Rajesh Jayaram、Cameron Musco、Christopher Musco 和 Archan Ray。通过显式和隐式消去改进谱密度估计。arXiv 预印本 arXiv:2410.21690，2024。[4] Mitchell Black、Zhengchao Wan、Amir Nayyeri 和 Yusu Wang。从有效电阻的角度理解图神经网络中的过度压缩。发表于国际机器学习会议，第 2528–2547 页。机器学习研究论文集，2023。[5] Francesco Bonchi、Pooya Esfandiar、David F Gleich、Chen Greif 和 Laks VS Lakshmanan。用于成对及按列计算往返时间和 Katz 分数的快速矩阵计算。Internet Mathematics，8(1-2)：73–112，2012。[6] Dongrun Cai、Xue Chen 和 Pan Peng。非扩展图中的有效电阻。arXiv 预印本 arXiv:2307.01218，2023。[7] Tyler Chen 和 Thomas Trogdon。具有正则谱分布的矩阵上 Lanczos 算法的稳定性。ArXiv，abs/2302.14842，2023。[8] Tyler Chen、Thomas Trogdon 和 Shashanka Ubaru。随机 Lanczos 求积用于谱近似的分析。发表于国际机器学习会议，第 1728–1739 页。机器学习研究论文集，2021。[9] Timothy Chu、Yu Gao、Richard Peng、Sushant Sachdeva、Saurabh Sawlani 和 Junxing Wang。通过短环分解实现图稀疏化、谱草图及更快的电阻计算。计算机科学基础研讨会，2020。[10] Fan Chung。热核作为图的 PageRank。美国国家科学院院刊，104(50)，2007。[11] Michael B Cohen、Rasmus Kyng、Gary L Miller、Jakub W Pachocki、Richard Peng、Anup B Rao 和 Shen Chen Xu。以近乎 m log1/2 n 的时间求解对称对角占优线性系统。发表于第四十六届 ACM 计算理论研讨会论文集，第 343–352 页，2014。[12] OpenStreetMap 贡献者。获取自 https://planet.osm.org 的 Planet 数据转储。https://www.openstreetmap.org.，2017。[13] Guanyu Cui、Hanzhi Wang 和 Zhewei Wei。混合时间至关重要：通过双向方法加速有效电阻估计。发表于第 31 届 ACM SIGKDD 知识发现与数据挖掘会议，KDD '25，2025。[14] Rajat Vadiraj Dwaraknath、Ishani Karmarkar 和 Aaron Sidford。迈向最优有效电阻估计。神经信息处理系统进展，36，2024。[15] Yuan Gao、Rasmus Kyng 和 Daniel A Spielman。通过近似消元稳健且实用地求解拉普拉斯方程。arXiv 预印本 arXiv:2303 00709，2023 。。[16] Gramoz Goranci、Monika Henzinger 和 Pan Peng。可分图上的动态有效电阻与近似舒尔补。发表于第 26 届欧洲算法研讨会（ESA），2018。[17] Arun Jambulapati 和 Aaron Sidford。用于拉普拉斯矩阵及其伪逆的高效 $\tilde{O}(n/\epsilon)$ 谱草图。发表于第二十九届 ACM-SIAM 离散算法研讨会论文集。SIAM，2018。[18] Rasmus Kyng 和 Sushant Sachdeva。拉普拉斯矩阵的近似高斯消元——快速、稀疏且简单。发表于 2016 年 IEEE 第 57 届计算机科学基础研讨会（FOCS），第 573–582 页。IEEE，2016。[19] Lawrence Li 和 Sushant Sachdeva。一种在扩展图中估计有效电阻并计算生成树数量的新方法。发表于 2023 年 ACM-SIAM 离散算法研讨会（SODA）论文集，第 2728–2745 页。SIAM，2023。[20] Meihao Liao、Rong-Hua Li、Qiangqiang Dai、Hongyang Chen、Hongchao Qin 和 Guoren Wang。高效电阻距离计算：基于地标的方法的优势。ACM数据管理会议录，2023。[21] Meihao Liao、Junjie Zhou、Rong-Hua Li、Qiangqiang Dai、Hongyang Chen 和 Guoren Wang。大型图上的高效且可证明的有效电阻计算：一种基于索引的方法。ACM数据管理会议录，2(3)：1–27，2024。[22] Peter Lofgren、Siddhartha Banerjee 和 Ashish Goel。个性化 PageRank 估计与搜索：一种双向方法。发表于网络搜索与数据挖掘会议，2016。[23] Peter Lofgren 和 Ashish Goel。面向目标节点的个性化 PageRank。arXiv 预印本 arXiv:1304.4658，2013。[24] Zenan Lu、Xiaotian Zhou 和 Zhongzhi Zhang。图上电阻直径的快速估计与优化。发表于 2025 年 ACM 万维网会议论文集，WWW '25，第 2391–2401 页，2025。[25] Aleksander Madry。利用增广电流计算最大流。发表于 2016 年 IEEE 第 57 届计算机科学基础研讨会（FOCS），第 593–602 页。IEEE，2016。[26] Gérard Meurant 和 Zdeněk Strakoš。有限精度算术中的 Lanczos 和共轭梯度算法。Acta Numerica，15:471–542，2006。[27] Gérard Meurant 和 Zdeněk Strakoš。有限精度算术中的 Lanczos 和共轭梯度算法。Acta Numerica，2006。[28] Moshe Morgenstern。对每个素数幂 q，都存在 q+ 1 正则 Ramanujan 图，且有显式构造。组合理论杂志B辑，62(1):44–62，1994。[29] Cameron Musco、Christopher Musco 和 Aaron Sidford。矩阵函数近似的 Lanczos 方法稳定性。载于 SODA。SIAM，2018。[30] Lorenzo Orecchia、Sushant Sachdeva 和 Nisheeth K Vishnoi。指数函数近似、Lanczos 方法及一种用于平衡分隔集的 o (m) 时间谱算法。载于 STOC，2012。[31] Lawrence Page、Sergey Brin、Rajeev Motwani 和 Terry Winograd。PageRank 引文排名：为网络带来秩序。技术报告，斯坦福信息实验室，1999。[32] Chris C Paige。对称特征值问题中 Lanczos 算法的精度和有效性。Linear Algebra and its Applications，34:235–258，1980。[33] Pan Peng、Daniel Lopatta、Yuichi Yoshida 和 Gramoz Goranci。用于估计有效电阻的局部算法。载于第 27 届 ACM SIGKDD知识发现与数据挖掘会议论文集，第 1329–1338 页，2021。[34] Dana Ron。用于近似图参数的次线性时间算法。载于《计算与软件科学：现状与展望》，第 105–122 页。施普林格，2019。[35] Sushant Sachdeva、Nisheeth K Vishnoi 等。通过逼近理论实现更快的算法。Foundations and Trends® in Theoretical Computer Science，9(2)，2014。[36] Purnamrita Sarkar、Andrew W. Moore 和 Amit Prakash。大型图中的快速增量式邻近搜索。载于第 25 届国际机器学习会议（ICML）论文集，2008。[37] Jieming Shi、Nikos Mamoulis、Dingming Wu 和 David W. Cheung。地理社交网络中基于密度的地点聚类。载于 2014 年 ACM SIGMOD 数据管理国际会议（ SIGMOD ）论文集，2014。[38] Pan Shi、Kun He、David Bindel 和 John E Hopcroft。用于社区检测的局部 Lanczos 谱近似。载于欧洲机器学习与数据库知识发现联合会议，第 651–667 页。施普林格，2017。[39] Ali Kemal Sinop、Lisa Fawcett、Sreenivas Gollapudi 和 Kostas Kollias。利用电流实现鲁棒路由。ACM Trans. Spatial Algorithms Syst.，2023。[40] Daniel A Spielman 和 Nikhil Srivastava。通过有效电阻进行图稀疏化。载于第 40 届 ACM 计算理论年度研讨会论文集，第 563–568 页，2008。[41] Anton Tsitsulin、Marina Munkhoeva 和 Bryan Perozzi。近似时只需 SLAQ：适用于网络规模图的精确谱距离。载于 2020 年万维网会议论文集，第 2697–2703 页，2020。[42] Jan van den Brand、Yu Gao、Arun Jambulapati、Yin Tat Lee、Yang P Liu、Richard Peng 和 Aaron Sidford。通过改进动态谱顶点稀疏器实现更快的最大流。载于第 54 届 ACM SIGACT 计算理论年度研讨会论文集，第 543–556 页，2022。[43] Hanzhi Wang、Mingguo He、Zhewei Wei、Sibo Wang、Ye Yuan、Xiaoyong Du 和 Ji-Rong Wen。近似图传播。载于 KDD，2021。[44] Hanzhi Wang 和 Zhewei Wei。在$\tilde{O}(mind_t, \sqrt{m})$时间内估计单节点 PageRank。Proc. VLDB Endow.，2023。[45] Hanzhi Wang、Zhewei Wei、Junhao Gan、Ye Yuan、Xiaoyong Du 和 Ji-Rong Wen。用于个性化PageRank 的基于边的局部推送。arXiv 预印本 arXiv:2203 07937，2022。。[46] Sibo Wang、Renchi Yang、Xiaokui Xiao、Zhewei Wei 和 Yin Yang。Fora：简单有效的单源个性化PageRank近似算法。载于 KDD，第 505–514 页，2017。[47] Virginia Williams。比 Coppersmith–Winograd 更快地进行矩阵乘法。第 887–898 页，05 2012。[48] Hao Wu、Junhao Gan、Zhewei Wei 和 Rui Zhang。统一全局方法与局部方法：一种结合前向推送的高效幂迭代。载于 SIGMOD，2021。[49] Renchi Yang 和 Jing Tang。成对有效电阻的高效估计。ACM数据管理会议录，1(1):1–27，2023。[50] Renchi Yang、Xiaokui Xiao、Zhewei Wei、Sourav S Bhowmick、Jun Zhao 和 Rong-Hua Li。用于局部聚类的热核PageRank高效估计。载于 SIGMOD，2019。[51] Yichun Yang、Rong-Hua Li、Meihao Liao 和 Guoren Wang。图上有效电阻计算的改进算法。COLT，2025。[52] Yichun Yang、Longlong Lin、Rong-Hua Li、Meihao Liao 和 Guoren Wang。大型图上的电阻距离计算：理论与实践均高效。https://anonymous.4open.science/r/LanczosPush-CCDC/，2025。[53] Yunfeng Yu、Longlong Lin、Qiyu Liu、Zeli Wang、Xi Ou 和 Tao Jia。GSD-GNN：适用于解耦图神经网络的通用且可扩展算法。载于 ICMR，2024。

# A 省略证明

## A.1 公式 1 的证明

定理 A.1。 （重述公式 *1*）

$$r_{\mathcal{G}}(s,t) = (\mathbf{e}_s - \mathbf{e}_t)^T \mathbf{L}^\dagger (\mathbf{e}_s - \mathbf{e}_t) = \frac{1}{2}(\mathbf{e}_s - \mathbf{e}_t)^T \sum_{k=0}^{+\infty} \mathbf{D}^{-1} \left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)^k (\mathbf{e}_s - \mathbf{e}_t)$$

证明。我们采用与 [33] 中引理 4.3 类似的论证。首先注意到 $\mathbf{L} = \mathbf{D} - \mathbf{A} = \mathbf{D}^{1/2}(\mathbf{I} - \mathbf{A})\mathbf{D}^{1/2}$，其中 A 是归一化邻接矩阵。因此，我们有：

$$\begin{aligned}
\mathbf{L}^\dagger &= \mathbf{D}^{-1/2}(\mathbf{I} - \mathcal{A})^\dagger \mathbf{D}^{-1/2} = \mathbf{D}^{-1/2} \sum_{j=2}^{n} \frac{1}{1-\lambda_j(\mathcal{A})} \mathbf{u}_j \mathbf{u}_j^T \mathbf{D}^{-1/2} \\
&= \frac{1}{2}\mathbf{D}^{-1/2} \sum_{j=2}^{n} \sum_{l=0}^{\infty} \left(\frac{1}{2} + \frac{1}{2}\lambda_j(\mathcal{A})\right)^l \mathbf{u}_j \mathbf{u}_j^T \mathbf{D}^{-1/2} \\
&= \frac{1}{2}\mathbf{D}^{-1/2} \sum_{l=0}^{\infty} \sum_{j=2}^{n} \left(\frac{1}{2} + \frac{1}{2}\lambda_j(\mathcal{A})\right)^l \mathbf{u}_j \mathbf{u}_j^T \mathbf{D}^{-1/2} \\
&= \frac{1}{2}\mathbf{D}^{-1/2} \sum_{l=0}^{\infty} \left(\left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathcal{A}\right)^l - \mathbf{u}_1 \mathbf{u}_1^T\right) \mathbf{D}^{-1/2}
\end{aligned}$$

其中 $\mathbf{u}_j$ 表示 A 对应于特征值 $\lambda_j(\mathrm{A})$ 的特征向量。接下来，我们注意到，对于任意 $s,t \in \mathrm{V}$，都有 $\mathbf{u}_1 = \mathbf{D}^{1/2}\mathbf{1} \perp \mathbf{D}^{-1/2}(\mathbf{e}_s - \mathbf{e}_t)$，其中 $\mathbf{1}$ 是全 1 向量。因此，

$$\begin{aligned}
r_{\mathcal{G}}(s,t) &= (\mathbf{e}_s - \mathbf{e}_t)^T \mathbf{L}^\dagger (\mathbf{e}_s - \mathbf{e}_t) \\
&= \frac{1}{2}(\mathbf{e}_s - \mathbf{e}_t)^T \mathbf{D}^{-1/2} \sum_{l=0}^{\infty} \left(\left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathcal{A}\right)^l - \mathbf{u}_1 \mathbf{u}_1^T\right) \mathbf{D}^{-1/2} (\mathbf{e}_s - \mathbf{e}_t) \\
&= \frac{1}{2}(\mathbf{e}_s - \mathbf{e}_t)^T \mathbf{D}^{-1/2} \sum_{l=0}^{+\infty} \left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathcal{A}\right)^l \mathbf{D}^{-1/2} (\mathbf{e}_s - \mathbf{e}_t) \\
&= \frac{1}{2}(\mathbf{e}_s - \mathbf{e}_t)^T \mathbf{D}^{-1} \sum_{l=0}^{+\infty} \left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)^l (\mathbf{e}_s - \mathbf{e}_t)
\end{aligned}$$

证明至此完成。 □

## A.2 定理 2.4 的证明

定理 A.2. （重述定理 *2.4*）当设置 $l = 2\kappa \log \frac{\kappa}{\epsilon}$ 时，近似误差满足 $|r_{\mathcal{G}}(s,t) - \bar{r}_{\mathcal{G}}(s,t)| \le \epsilon$。此外，幂法的时间复杂度为 $O(lm) = O(\kappa m \log \frac{\kappa}{\epsilon})$。

证明。由于 $\bar{r}_{\mathcal{G}}(s,t) = \frac{1}{2}(\mathbf{e}_s - \mathbf{e}_t)^T \sum_{k=0}^{l} \mathbf{D}^{-1} \left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)^k (\mathbf{e}_s - \mathbf{e}_t)$ 是幂法的输出，因此近似误差满足：

$$\begin{aligned}
&|\bar{r}_{\mathcal{G}}(s,t) - r_{\mathcal{G}}(s,t)| \\
&= \left|\frac{1}{2}(\mathbf{e}_s - \mathbf{e}_t)^T \mathbf{D}^{-1} \sum_{l=L+1}^{\infty} \left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)^l (\mathbf{e}_s - \mathbf{e}_t)\right| \\
&= \left|\frac{1}{2}(\mathbf{e}_s - \mathbf{e}_t)^T \mathbf{D}^{-1/2} \sum_{l=L+1}^{\infty} \left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathcal{A}\right)^l \mathbf{D}^{-1/2} (\mathbf{e}_s - \mathbf{e}_t)\right| \\
&= \left|\frac{1}{2}(\mathbf{e}_s - \mathbf{e}_t)^T \mathbf{D}^{-1/2} \sum_{l=L+1}^{\infty} \sum_{j=2}^{n} \left(\frac{1}{2} + \frac{1}{2}\lambda_j(\mathcal{A})\right)^l \mathbf{u}_j \mathbf{u}_j^T \mathbf{D}^{-1/2} (\mathbf{e}_s - \mathbf{e}_t)\right|
\end{aligned}$$

其中，$\mathbf{u}_j$ 表示与特征值 $\lambda_j(\mathrm{A})$ 对应的 A 的特征向量。接下来，我们记 $\alpha_j = \mathbf{u}_j^T \mathbf{D}^{-1/2}(\mathbf{e}_s - \mathbf{e}_t)$，以及 $\alpha = (\alpha_1, \ldots, \alpha_n)^T = (\mathbf{u}_1, \ldots, \mathbf{u}_n)^T \mathbf{D}^{-1/2}(\mathbf{e}_s - \mathbf{e}_t)$。因此，我们可以对其进行界定

$$\|\alpha\|_2 = \|(\mathbf{u}_1, \ldots, \mathbf{u}_n)\mathbf{D}^{-1/2}(\mathbf{e}_s - \mathbf{e}_t)\|_2 \le \|\mathbf{D}^{-1/2}(\mathbf{e}_s - \mathbf{e}_t)\|_2 \le \sqrt{2}$$

因此 $\sum_{j=1}^{n} \alpha_j^2 \le 2$。所以，

$$\begin{aligned}
|\bar{r}_{\mathcal{G}}(s,t) - r_{\mathcal{G}}(s,t)| &= \left|\frac{1}{2} \sum_{l=L+1}^{\infty} \sum_{j=2}^{n} \left(\frac{1}{2} + \frac{1}{2}\lambda_j(\mathcal{A})\right)^l \alpha_j^2\right| \\
&\le \sum_{l=L+1}^{\infty} \left(\frac{1}{2} + \frac{1}{2}\lambda_2(\mathcal{A})\right)^l
\end{aligned}$$

其中 $\lambda_2(\mathrm{A})$ 是 A 的第二大特征值。根据归一化拉普拉斯矩阵的特征值性质，$\lambda_2 = 1 - \lambda_2(\mathrm{L})$ (A)。因此，

$$\begin{aligned}
|\bar{r}_{\mathcal{G}}(s,t) - r_{\mathcal{G}}(s,t)| &\le \sum_{l=L+1}^{\infty} \left(1 - \frac{1}{2}\lambda_2(\mathcal{L})\right)^l \le \left(1 - \frac{1}{2}\lambda_2(\mathcal{L})\right)^L \frac{2}{\lambda_2(\mathcal{L})} \\
&\le \left(1 - \frac{1}{2}\lambda_2(\mathcal{L})\right)^L \kappa(\mathcal{L}) \le \epsilon
\end{aligned}$$

设置 $L = \log_{\kappa}(\mathcal{L}) / \log\left(1 - \frac{1}{2}\lambda_2(\mathrm{L})\right)$ 时，最后一个不等式成立。我们注意到 $\log\left(1 - \frac{1}{2}\lambda_2(\mathrm{L})\right) \le -\frac{1}{2}\lambda_2(\mathrm{L})$ 和 $\kappa(\mathrm{L}) \le \lambda_2^2(\mathrm{L})$ </style>这一事实，因此设置 (L) log 就足够了。最后，我们注意到，幂法的每次迭代都需要计算矩阵-向量乘积 $\mathrm{I} + \frac{1}{2}\mathrm{P})\mathbf{r}$，这需要 $O(m)$ 次运算。因此，幂法的总时间复杂度为 $O(\bar{l}m) = O(\kappa m \log \frac{\kappa}{\epsilon})$。□

## A.3 引理 3.1 的证明

引理 A.3. （重述引理 *3.1*）给定 $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_k]$ 是张成子空间 $\mathcal{K}$ $span_{1, \ldots, k}(\mathbf{v}_1) = \mathcal{K}$ $(\mathcal{A})\langle \mathbf{v}\mathcal{A}\mathbf{v}, \mathcal{A}\mathbf{v}\rangle$ 的 $n \times k$ 维正交矩阵，且 $\mathbf{T} = \mathbf{V}\mathcal{A}\mathbf{V}$。那么，对于任意次数至多为 $k$ 的多项式 $p_k$，有 $p_k(\mathcal{A})\mathbf{v}_1 = \mathbf{V}p_k(\mathbf{T})\mathbf{V}^T\mathbf{v}_1$。

证明。由于 $\mathrm{A}^i\mathbf{v}_1 \in \mathrm{K}_k(\mathbf{v}_1, \mathrm{A})$ 对于 $i \le k$ 成立，且 $\mathbf{V}\mathbf{V}^T$ 是投影到子空间 $\mathrm{K}_k(\mathbf{v}_1, \mathrm{A})$ 上的投影矩阵，因此 $\mathrm{A}^i\mathbf{v}_1 = \mathbf{V}\mathbf{V}^T\mathrm{A}^i\mathbf{v}_1$。因此

$$\begin{aligned}
\mathcal{A}^i\mathbf{v}_1 &= (\mathbf{V}\mathbf{V}^T)\mathcal{A}(\mathbf{V}\mathbf{V}^T)\mathcal{A}\ldots\mathcal{A}(\mathbf{V}\mathbf{V}^T)\mathbf{v}_1 \\
&= \mathbf{V}(\mathbf{V}^T\mathcal{A}\mathbf{V})\ldots(\mathbf{V}^T\mathcal{A}\mathbf{V})\mathbf{V}^T\mathbf{v}_1 = \mathbf{V}\mathbf{T}^i\mathbf{V}^T\mathbf{v}_1.
\end{aligned}$$

对任意 $i \le k$ 均成立。因此，对于次数至多为 $k$ 的任意多项式 $p_k$，$p_k(\mathrm{A})\mathbf{v}_1 = \mathbf{V}p_k(\mathbf{T})\mathbf{V}^T\mathbf{v}_1$。□

## A.4 引理 3.2 的证明

引理 A.4。 （引理 *3.2* 的重述）设引理 *3.1* 中定义的 $\mathbf{V} \in \mathbb{R}^{n \times k}$、$\mathbf{T} \in \mathbb{R}^{k \times k}$。设 $\mathcal{P}_k$ 为所有度为 $\le k$ 的多项式的集合。则误差满足：

$$\begin{aligned}
&|\mathbf{v}_1^T\mathbf{V}(\mathbf{I} - \mathbf{T})^{-1}\mathbf{V}^T\mathbf{v}_1 - \mathbf{v}_1^T(\mathbf{I} - \mathcal{A})^\dagger\mathbf{v}_1| \\
&\le 2 \min_{p \in \mathcal{P}_k} \max_{x \in [\lambda_{min}(\mathcal{A}), \lambda_2(\mathcal{A})]} |1/(1-x) - p(x)|
\end{aligned}$$

其中，$\lambda_{min}(\mathcal{A})$ 和 $\lambda_2(\mathcal{A})$ 分别表示 $\mathcal{A}$ 的最小特征值和第二大特征值。

证明。注意，对于任意次数为 $\leq k$ 的多项式 $p$，以下不等式成立：

$$|\mathbf{v}_1^T\mathbf{V}(\mathbf{I}-\mathbf{T})^{-1}\mathbf{V}^T\mathbf{v}_1 - \mathbf{v}_1^T(\mathbf{I}-\mathcal{A})^\dagger\mathbf{v}_1| = |\mathbf{e}_1^T(\mathbf{I}-\mathbf{T})^{-1}\mathbf{e}_1 - \mathbf{v}_1^T(\mathbf{I}-\mathcal{A})^\dagger\mathbf{v}_1|$$
$$\leq |\mathbf{v}_1^T(\mathbf{I}-\mathcal{A})^\dagger\mathbf{v}_1 - \mathbf{v}_1^T p(\mathcal{A})\mathbf{v}_1| + |\mathbf{v}_1^T p(\mathcal{A})\mathbf{v}_1 - \mathbf{e}_1^T p(\mathbf{T})\mathbf{e}_1|$$
$$+ |\mathbf{e}_1^T p(\mathbf{T})\mathbf{e}_1 - \mathbf{e}_1^T(\mathbf{I}-\mathbf{T})^{-1}\mathbf{e}_1| \tag{6}$$

对于式 (6) 右侧的第二项，根据引理 3 1，该项为 0。现在我们来界定上述不等式的第一项和第三项。注意，A 的特征分解为 A $=\sum_{i=1}^n \lambda_i \mathbf{u}_i\mathbf{u}_i^T$，其中 $\lambda_i$ 是 A 的第 $i$ 大特征值，$\mathbf{u}_i$ 是对应的特征向量。根据伪逆的定义，$(\mathbf{I}-\mathrm{A})^\dagger$ 的特征分解为 $(\mathbf{I}-\mathrm{A})^\dagger = \sum_{i=2}^n \frac{1}{1-\lambda_i}\mathbf{u}_i\mathbf{u}_i^T$，而 $p(\mathrm{A})$ 的特征分解为 $p(\mathrm{A}) = \sum_{i=1}^n p(\lambda_i)\mathbf{u}_i\mathbf{u}_i^T$。由于第一个特征向量 $\mathbf{u}_1 = \mathbf{D}^{1/2}\mathbf{1} \perp \mathbf{v}_1$，，因此式 (6) 第二行的第一项可表示如下：

$$|\mathbf{v}_1^T(\mathbf{I}-\mathcal{A})^\dagger\mathbf{v}_1 - \mathbf{v}_1^T p(\mathcal{A})\mathbf{v}_1| = |\sum_{i=2}^n (\frac{1}{1-\lambda_i} - p(\lambda_i))(\mathbf{v}_1^T\mathbf{u}_i)^2|$$

接下来，我们定义另一个向量 $c = \mathbf{v}_1^T(\mathbf{u}_1,\ldots,\mathbf{u}_n)$。注意到 $\|c\|_2 = \|\mathbf{v}_1^T\mathbf{u}_1 \ldots \mathbf{v}_1^T\mathbf{u}_n\|_2 = \|\mathbf{v}_1^T\mathbf{U}\|_2 = 1$ $\sum_{i=1}^n c_i^2 = \mathbf{v}_1^T(\mathbf{u}_1,\ldots,\mathbf{u}_n)\mathbf{v}_1^T$，因此 $\sum_{i=1}^n (\mathbf{v}_1^T\mathbf{u}_i)^2 = 1$。
由此，我们可以对上述等式进行界定：

$$|\sum_{i=2}^n (\frac{1}{1-\lambda_i} - p(\lambda_i))(\mathbf{v}_1^T\mathbf{u}_i)^2| \leq \max_{x\in[\lambda_{min}(\mathcal{A}),\lambda_2(\mathcal{A})]} |1/(1-x) - p(x)|$$

对$\mathbf{T}$采用类似的论证，我们得到：

$$|\mathbf{e}_1^T p(\mathbf{T})\mathbf{e}_1 - \mathbf{e}_1^T(\mathbf{I}-\mathbf{T})^{-1}\mathbf{e}_1| \leq \max_{x\in[\lambda_{min}(\mathbf{T}),\lambda_{max}(\mathbf{T})]} |1/(1-x) - p(x)|$$

最后，我们注意到$\mathbf{T}$是 A 在子空间 $\mathrm{K}_k(\mathrm{A},\mathbf{v}_1) = span\mathbf{v}_1, \mathrm{A}\mathbf{v}_1, \ldots, \mathrm{A}^k\mathbf{v}_1$ 上的投影，其中$\mathbf{v}_1$与最大特征值 $\lambda_1(\mathrm{A})$ 对应的特征子空间正交。因此，$\lambda_{min}(\mathbf{T}) \geq \lambda_{min}(\mathrm{A})$，且 $\lambda_{max}(\mathbf{T}) \leq \lambda_2(\mathrm{A})$。由此可得：

$$\max_{x\in[\lambda_{min}(\mathbf{T}),\lambda_{max}(\mathbf{T})]} |1/(1-x) - p(x)| \leq \max_{x\in[\lambda_{min}(\mathcal{A}),\lambda_2(\mathcal{A})]} |1/(1-x) - p(x)|$$

综合这些不等式，我们得到引理 3.2 的结论：

$$|\mathbf{v}_1^T\mathbf{V}(\mathbf{I}-\mathbf{T})^{-1}\mathbf{V}^T\mathbf{v}_1 - \mathbf{v}_1^T(\mathbf{I}-\mathcal{A})^\dagger\mathbf{v}_1| \leq 2\min_{p\in\mathcal{P}_k}\max_{x\in[\lambda_{min}(\mathcal{A}),\lambda_2(\mathcal{A})]} |1/(1-x) - p(x)|$$

至此，证明完成 .

□

## A.5 定理 3.4 的证明

定理 A.5. （重述定理 *3.4*）设 $\hat{r}_\mathcal{G}(s,t)$ 为 *Lanczos* 迭代（算法 *3*）的输出。则误差满足

$$|\hat{r}_\mathcal{G}(s,t) - r_\mathcal{G}(s,t)| \leq \epsilon$$

当迭代次数设为 $k = O(\sqrt{\kappa}\log\frac{\kappa}{\epsilon})$ 时。此外，算法 *3* 的时间复杂度为 $O(km + k^{2.37}) = O(\sqrt{\kappa}m\log\frac{\kappa}{\epsilon})$。

We observe that $\hat{r}_\mathcal{G}(s,t)$ output by Algorithm 3 satisfies $\hat{r}_\mathcal{G}(s,t) = \left(\frac{1}{d_s} + \frac{1}{d_t}\right)\mathbf{e}_1^T(\mathbf{I}-\mathbf{T})^{-1}\mathbf{e}_1$ and $r_\mathcal{G}(s,t) = \left(\frac{1}{d_s} + \frac{1}{d_t}\right)\mathbf{v}_1(\mathbf{I}-\mathcal{A})^\dagger\mathbf{v}_1$, so 根据引理 3.2，我们有：

$$|\hat{r}_\mathcal{G}(s,t) - r_\mathcal{G}(s,t)| \leq \left(\frac{1}{d_s} + \frac{1}{d_t}\right) 2\min_{p\in\mathcal{P}_k}\max_{x\in[\lambda_{min}(\mathcal{A}),\lambda_2(\mathcal{A})]} |1/(1-x) - p(x)|$$

因此，剩下的工作就是对上述不等式右侧进行界定。为此，我们证明存在一个次数至多为 $k = O(\sqrt{\kappa}\log\frac{\kappa}{\epsilon})$ 的多项式，它在范围 $[\lambda_{min}(\mathrm{A})\ \lambda_2(\mathrm{A})]$ 内以 $\epsilon$ 逼近 $1/(1-x)$。

定理 A.6. 对于任意给定的无穷次可微函数 $f$、参数 $\epsilon \in (0,1)$、条件数为 $\kappa$ 的矩阵 $\mathcal{A} \in \mathbb{R}^{n\times n}$，令 $k = O(\sqrt{\kappa}\log\frac{\kappa}{\epsilon})$。存在一个次数为 $\leq k$ 的多项式 $p_k$，满足：

$$\max_{x\in[\lambda_{min}(\mathcal{A}),\lambda_2(\mathcal{A})]} |1/(1-x) - p_k(x)| < \frac{\epsilon}{2}$$

具体而言，$p_k(x) = \sum_{l=0}^k c_l T_l(x)$ 是切比雪夫多项式 $T_l(x)$ 的线性组合，系数为 $c_l > 0$，且 $\sum_l c_l \leq \kappa\log(\frac{\kappa}{\epsilon})$。

我们注意到，定理 3.4 可由定理 A.6 直接推出。事实上，根据定理 A.6，令迭代次数为 $k = O(\sqrt{\kappa}\log\frac{\kappa}{\epsilon})$，即可达到 $\epsilon$ 的绝对值保证。每次迭代需要 $O(m)$ 次运算，最后还需要额外花费 $O(k^{2.37})$ 时间来计算 $\mathbf{e}_1^T(\mathbf{I}-\mathbf{T})^{-1}\mathbf{e}_1$。因此，总时间复杂度至多为 $O(km + k^{2.37}) = O(\sqrt{\kappa}m\log\frac{\kappa}{\epsilon})$。剩余部分用于证明定理 A.6。为证明定理 A.6，我们使用以下第一类切比雪夫多项式展开。回顾一下，第一类切比雪夫多项式 $\{T_l(x)\}_{l\geq 0}$ 由以下递推关系生成：

$$T_0(x) = 1, T_1(x) = x$$
$$T_{l+1}(x) = 2xT_l(x) - T_{l-1}(x), \quad l \geq 1 \tag{7}$$

引理 A.7. 对于任意 $k \geq 0, x^k = \sum_{l\in\mathbb{Z}} q_k(l)T_{|l|}(x)$，其中 $q_k(l)$ 表示从 0 出发的简单随机游走在经过 $k$ 步后位于 $l$ 的概率。

证明。引理 A.7 可通过计算得出。设 $x = \cos\theta \text{and} w = e^{i\theta}$，我们有：

$$x^k = [(w + w^{-1})/2]^k = \sum_{l\in\mathbb{Z}} q_k(l)w^l$$
$$= \sum_{l\in\mathbb{Z}} q_k(l)(w^l + w^{-l})/2 = \sum_{l\in\mathbb{Z}} q_k(l)\cos(l\theta)$$
$$= \sum_{l\in\mathbb{Z}} q_k(l)T_{|l|}(x)$$

这就证明了引理 A.7。 □

接下来，我们使用引理 A.7 来证明定理 A.6。证明思路如下：首先使用泰勒展开来表示 $1/(1-x)$：$1/(1-x) = \frac{1}{2}\sum_{t=0}^\infty (\frac{1}{2} + \frac{1}{2}x)^t$。接下来，我们选择一个足够大的截断步数 $L$，并考虑 $L$ 步截断多项式 $\frac{1}{2}\sum_{t=0}^L (\frac{1}{2} + \frac{1}{2}x)^t$。我们证明，当设定 $L = \kappa\log(\frac{\kappa}{\epsilon})$ 时，$\frac{1}{2}\sum_{t=0}^L (\frac{1}{2} + \frac{1}{2}x)^t$ 足够接近 $1/(1-x)$。然后，我们先用 $k$ 个切比雪夫多项式（即使用引理 A.7）来逼近 $L$ 次多项式 $\frac{1}{2}\sum_{t=0}^L (\frac{1}{2} + \frac{1}{2}x)^t$，并证明设定 $k = \sqrt{L}\log\frac{\kappa}{\epsilon}$ 就足够了。为此，我们有以下两个断言：

命题 $\frac{\epsilon}{4}$ $\max_{x\in[\lambda_{min}\ \lambda_2]} |1 - x - \frac{1}{2}\sum_{t=0}^L (\frac{1}{2} + \frac{1}{2}x)^t| < \max(A)(A)/$ $L = O(\kappa\log\frac{\kappa}{\epsilon})$ ( )( )取值为 ( log( )).15

证明。根据泰勒展开的性质，我们知道 $1/(1-x) - \frac{1}{2}\sum_{t=0}^{L}(\frac{1}{2}+\frac{1}{2}x)^t = \frac{1}{2}\sum_{t=L+1}^{\infty}(\frac{1}{2}+\frac{1}{2}x)^t$。接下来，对于 $x \in [\lambda_{min}(\mathbf{A}), \lambda_2(\mathbf{A})]$，我们可以将 $\frac{1}{2}\sum_{t=L+1}^{\infty}(\frac{1}{2}+\frac{1}{2}x)^t = (\frac{1}{2}+\frac{1}{2}x)^{L}\cdot\frac{1}{2}\sum_{t=0}^{\infty}(\frac{1}{2}+\frac{1}{2}x)^t \le (\frac{1}{2}+\frac{1}{2}\lambda_2(\mathcal{A}))^L \cdot \frac{1}{1-\lambda_2(\mathcal{A})} \le \frac{\epsilon}{4}$ 进行界定。最后一个不等式通过令 $L = \kappa\log(\frac{\kappa}{\epsilon})$ 得出。这证明了命题 1。□

命题 2. 存在一个次数至多为 $k$的多项式 $p_k$，使得 $\max_{x\in[-1,1]}|\frac{1}{2}\sum_{t=0}^{L}(\frac{1}{2}+\frac{1}{2}x)^t - p_k(x)| < \frac{\epsilon}{4}$，其中令 $k = \sqrt{2L\log(\frac{8L}{\epsilon})}$。

证明。首先，对于每个矩 $x^t$，我们使用引理 A.7：$x^t = \sum_{l\in\mathbb{Z}} q_t(l)T_{|l|}(x)$。根据阿祖马-霍夫丁不等式，$\sum_{|l|\ge k} q_t(l) \le 2e^{-k^2/2t}$。令 $k = \sqrt{2L\log(\frac{8L}{\epsilon})}$，则 $\sum_{|l|\ge k} q_t(l) \le \exp(\log(\frac{\epsilon}{4L}))$。我们选择 $\log(\cdot)$，于是 $(\cdot)$。因此，我们定义次数为 的切比雪夫截断多项式：$q_t(x) = \sum_{|l|\le k} q_t(l)T_{|l|}(x)$，有：

$$\max_{x\in[-1,1]}|x^t - q_t(x)| \le \sum_{|l|\ge k} q_t(l) \le \frac{\epsilon}{4L}$$

那么，我们定义度为 $k$ 的多项式 $p_k$ 为 $p_k(x) = \sum_{t=0}^{L} q_t(x)$，有：

$$\begin{aligned}\max_{x\in[-1,1]}|\frac{1}{2}\sum_{t=0}^{L}(\frac{1}{2}+\frac{1}{2}x)^t - p_k(x)| &\le \sum_{t=0}^{L}\max_{x\in[-1,1]}|x^t - q_t(x)| \\ &\le L\frac{\epsilon}{4L} = \frac{\epsilon}{4}\end{aligned} \tag{8}$$

这证明了命题 2 . □

根据命题 1 和命题 2，我们知道，$1/(1-x)$ 可以用多项式 $p_k$ 近似，误差至多为 $\frac{\epsilon}{2}$，且 $k = \sqrt{2L\log(\frac{8L}{\epsilon})} = O(\sqrt{\kappa\log\frac{\kappa}{\epsilon}})$。最后，根据 $p_k(x) = \sum_{t=0}^{L} q_t(x)$，而 $q_t(x)$ 是关于$t \le L$的切比雪夫多项式的线性组合，因此 $p_k(x)$ 是切比雪夫多项式的线性组合。我们将此扩张性 $p_k(x) = \sum_{t=0}^{L} q_t(x) = \sum_{l=0}^{k} c_l T_l(x)$ 记为系数为 $c_l$ 的形式。由于 $q_t(l)$ 是一个概率分布（即，$\sum_{l\in\mathbb{Z}} q_t(l) = 1$且 $q_t(l) \ge 0$），因此系数 $c_l \ge 0$ 的和至多为：$\sum_{l=0}^{k} c_l \le \sum_{l=0}^{L}\sum_{l\in\mathbb{Z}} q_t(l) = L = \kappa\log(\frac{\kappa}{\epsilon})$。

## A.6 定理 4.3 的证明

定理 A.8. （重述定理 *4.3*）给定 $\epsilon' \in (0,1)$，设置迭代次数 $k = \sqrt{\kappa}\log\frac{\kappa}{\epsilon'}$、参数 $\epsilon = \tilde{\Omega}(\frac{\epsilon\sqrt{d}}{\kappa^{2.25}C_1'})$，算法 *4* 输出的近似值 $r'_{\mathcal{G}}(s,t)$与精确 *ER* 值 $r_{\mathcal{G}}(s,t)$之间的误差可界定为：

$$|r_{\mathcal{G}}(s,t) - r'_{\mathcal{G}}(s,t)| \le \epsilon'$$

进行理论分析时，回顾一下我们需要假设 1。现在我们转向定理 4.3 的证明。与定理 3.4 证明中的公式 6 类似，我们将误差分成三项：

$$\begin{aligned}|\hat{r}_{\mathcal{G}}(s,t) - r_{\mathcal{G}}(s,t)|/(\frac{1}{d_s}+\frac{1}{d_t}) &\le |\mathbf{v}_1^T(\mathbf{I}-\mathcal{A})^{\dagger}\mathbf{v}_1 - \mathbf{v}_1^T p(\mathcal{A})\mathbf{v}_1| \\ &+ |\mathbf{v}_1^T p(\mathcal{A})\mathbf{v}_1 - \hat{\mathbf{v}}_1^T\hat{\mathbf{V}}p(\hat{\mathbf{T}})\mathbf{e}_1| \\ &+ |\hat{\mathbf{v}}_1^T\hat{\mathbf{V}}p(\hat{\mathbf{T}})\mathbf{e}_1 - \mathbf{e}_1^T(\mathbf{I}-\hat{\mathbf{T}})^{-1}\mathbf{e}_1|\end{aligned} \tag{9}$$

对于公式 9 右侧的第一项，按照引理 3.2 证明中的相同论证，通过将迭代次数设为 $k = O(\sqrt{\kappa}\log(\frac{\kappa}{\epsilon'}))$，其小于 $\epsilon/8$。对于第三项公式 9 右侧的第三项，根据假设 1 以及引理 3.2 证明中的相同论证，其也小于 $\epsilon/8$。因此我们得到误差界：

$$|\mathbf{v}_1^T(\mathbf{I}-\mathcal{A})^{\dagger}\mathbf{v}_1 - \hat{\mathbf{v}}_1^T\hat{\mathbf{V}}(\mathbf{I}-\hat{\mathbf{T}})^{-1}\mathbf{e}_1| \le |\mathbf{v}_1^T p(\mathcal{A})\mathbf{v}_1 - \hat{\mathbf{v}}_1^T\hat{\mathbf{V}}p(\hat{\mathbf{T}})\mathbf{e}_1| + \frac{\epsilon'}{4}$$

接下来，我们关注公式 9 中中间项 $|\mathbf{v}_1^T p(\mathbf{A})\mathbf{v}_1 - \hat{\mathbf{v}}_1^T\hat{\mathbf{V}}p(\hat{\mathbf{T}})\mathbf{e}_1|$的界。回顾一下， $p$是由引理 A.6 选取的多项式，它展开为 $p(x) = \sum_{l=0}^{k} c_l T_l(x)$，其中 $\{T_l(x)\}$ 是第一类切比雪夫多项式级数，系数为 $\sum_l |c_l| \le \tilde{O}(\kappa)$。因此，多项式偏差级数可进一步表示为

$$\mathbf{v}_1^T p(\mathcal{A})\mathbf{v}_1 - \mathbf{v}_1^T\hat{\mathbf{V}}p(\hat{\mathbf{T}})\mathbf{e}_1 = \sum_{l=0}^{k} c_l\mathbf{v}_1^T(\hat{\mathbf{V}}T_l(\hat{\mathbf{T}})\mathbf{e}_1 - T_l(\mathcal{A})\mathbf{v}_1) \tag{10}$$

我们将 $d_l = \hat{\mathbf{V}}T_l(\hat{\mathbf{T}})\mathbf{e}_1 - T_l(\mathbf{A})\mathbf{v}_1$ 定义为切比雪夫基 $T_l(x)$ 上的偏差约束。因此，误差项现在可以简单地表示为：

$$|\mathbf{v}_1^T p(\mathcal{A})\mathbf{v}_1 - \mathbf{v}_1^T\hat{\mathbf{V}}p(\hat{\mathbf{T}})\mathbf{e}_1| \le \sum_{l=0}^{k}|c_l||\mathbf{v}_1^T d_l|$$

现在我们关注 $d_l$ 的界。下一步，我们沿用分析 Lanczos 算法 [29] 稳定性时采用的相同策略。首先，通过与命题 3.3 比较，我们注意到，Lanczos Push 也满足类似的方程，但多了一个附加误差项。

命题 A.9。 *Lanczos Push* 算法（算法 *4*）计算出 $\hat{\mathbf{V}} \in \mathbb{R}^{n\times k}$、$\hat{\mathbf{T}} \in \mathbb{R}^{k\times k}$ 和一个附加向量 $\hat{\mathbf{v}}_{k+1}$，以及一个标量$\hat{\beta}_{k+1}$，使得：

$$\mathcal{A}\hat{\mathbf{V}} = \hat{\mathbf{V}}\hat{\mathbf{T}} + \hat{\beta}_{k+1}\hat{\mathbf{v}}_{k+1}\mathbf{e}_k^T + \mathbf{E}$$

其中，$\mathbf{E}$ 定义为误差矩阵，$\mathbf{E} = [\delta_1, \delta_2, ..., \delta_k]$，而$\delta_i \in \mathbb{R}^n$定义为 $\delta_i = (AMV(\mathcal{A}, \hat{\mathbf{v}}_i) - \mathcal{A}\hat{\mathbf{v}}_i) - \hat{\alpha}_i\hat{\mathbf{v}}_i|_{\mathcal{V}-S_i} - \hat{\beta}_i\hat{\mathbf{v}}_{i-1}|_{\mathcal{V}-S_{i-1}}$，对每个 $i \le k$ 均成立。

证明。回顾一下，在算法 4 的每次迭代中（即第 3–17 行），我们执行子集 Lanczos 递推：

$$\begin{aligned}\hat{\beta}_{i+1}\hat{\mathbf{v}}_{i+1} &= AMV(\mathcal{A}, \hat{\mathbf{v}}_i) - \hat{\alpha}_i\hat{\mathbf{v}}_i|_{S_i} - \hat{\beta}_i\hat{\mathbf{v}}_{i-1}|_{S_{i-1}} \\ &= (\mathcal{A} - \hat{\alpha}_i)\hat{\mathbf{v}}_i - \hat{\beta}_i\hat{\mathbf{v}}_{i-1} + \delta_i\end{aligned}$$

其中 $S_i = \{u \in \mathbf{V} : |\hat{\mathbf{v}}_i(u)| > \epsilon d_u\}$ 且 $\delta_i = (AMV(\mathbf{A}, \hat{\mathbf{v}}_i) - \mathbf{A}\hat{\mathbf{v}}_i) - \hat{\alpha}_i\hat{\mathbf{v}}_i|_{\mathcal{V}-S_i} - \hat{\beta}_i\hat{\mathbf{v}}_{i-1}|_{\mathcal{V}-S_{i-1}}$。由于该等式对任意 $\hat{i} \le k$ 均成立，根据$\mathbf{V}$、$\mathbf{T}$ 和$\mathbf{E}$的定义，这等价于：

$$\mathcal{A}\hat{\mathbf{V}} = \hat{\mathbf{V}}\hat{\mathbf{T}} + \hat{\beta}_{k+1}\hat{\mathbf{v}}_{k+1}\mathbf{e}_k^T + \mathbf{E}$$

证明至此完成 . □

接下来，我们证明，对于 $j \le l$，偏差 $d_l$ 可以表示为 $\xi_j$ 上第二类切比雪夫多项式的线性组合，其中 $\xi_j$定义为 $\xi_j = 2\mathbf{E}T_j(\hat{\mathbf{T}})\mathbf{e}_1$。在正式给出结果之前，我们先回顾一下，第二类切比雪夫多项式 $\{U_l(x)\}_{l\ge 0}$ 由以下递推关系生成：

$$\begin{aligned}U_0(x) = 1, U_1(x) = 2x \\ U_{l+1}(x) = 2xU_l(x) - U_{l-1}(x), \quad l \ge 1\end{aligned}$$

引理 *A.10*。设 $\hat{}V$、$\hat{}T$ 由 *Lanczos Push* 输出，$E$ 为命题 *A.9* 中定义的误差矩阵。对于每个 $\hat{}\hat{}$ 整数 $l \le k$，定义 $\xi_l = 2E T_l(\hat{}T)e_1$，并将 *Lanczos Push* 在 $T_l(A)$ 上施加约束所产生的偏差定义为 $d_l = VT_l(T)e_1 - T_l(A)v_1$。那么，对于每个 $l \le k$，$d_l$ 可以表示为 $d_l = -\frac{1}{2}U_{l-1}(A)\xi_1 - \sum_{j=2}^{l-1} U_{l-j}(A)\xi_j$，其中 $U_l(x)$ 是第二类切比雪夫多项式。

**引理 A.10 的证明**。这里的关键观察是，$d_l$ 满足三项递推关系，并且递推系数与切比雪夫多项式的相同。为使这一观察形式化，我们陈述以下事实：

Fact 1. $d_{l+1} = 2\mathcal{A}d_l - d_{l-1} - \xi_l$ *for* $l \le k-1$.

证明。根据 $d_{l+1}$ 的定义和命题 A.9，我们有：$d_{l+1} = \hat{}\mathbf{V}T_{l+1}(\hat{}\mathbf{T})\mathbf{e}_1 - T_{l+1}(\mathbf{A})\mathbf{v}_1$

$$\begin{aligned}
&= \hat{\mathbf{V}}(2\hat{\mathbf{T}}T_l(\hat{\mathbf{T}}) - T_{l-1}(\hat{\mathbf{T}}))\mathbf{e}_1 - (2\mathcal{A}T_l(\mathcal{A}) - T_{l-1}(\mathcal{A}))\mathbf{v}_1 \\
&= 2(\mathcal{A}\hat{\mathbf{V}} - \hat{\beta}_{k+1}\hat{\mathbf{v}}_{k+1}\mathbf{e}_k^T - \mathbf{E})T_l(\mathbf{T})\mathbf{e}_1 - 2\mathcal{A}T_k(\mathcal{A})\mathbf{v}_1 - d_{l-1} \\
&= 2\mathcal{A}d_l - d_{l-1} - 2\hat{\beta}_{k+1}\hat{\mathbf{v}}_{k+1}\mathbf{e}_k^T T_l(\mathbf{T})\mathbf{e}_1 - 2\mathbf{E}T_l(\mathbf{T})\mathbf{e}_1
\end{aligned}$$

我们注意到，根据定义，对于 $l \le k-1$ 和 $\xi_l = 2$，$\mathbf{e}_k^T T_l(\mathbf{T})\mathbf{e}_1 = 0$，并且 $d_{l+1} = 2\mathrm{A}d_l - d_{l-1} - \xi_l$。因此，对于 $l \le k-1$，我们有 $d_{l+1} = 2\mathrm{A}d_l - d_{l-1} - \xi_l$。这就完成了事实 1 的证明。□

下一个观察是，$d_l$ 的递推关系（即事实 1）恰好满足第二类切比雪夫多项式的递推关系。具体来说，我们声称，$d_l = -U_{l-1}(\mathrm{A})\xi_1 - \sum_{j=2}^{l} U_{l-j}(\mathrm{A})\xi_j$ 恰好满足事实 1 中所述的递推关系。

Fact 2. $d_l = -\frac{1}{2}U_{l-1}(\mathcal{A})\xi_1 - \sum_{j=2}^{l-1} U_{l-j}(\mathcal{A})\xi_j$ *for* $1 \le l \le k$.

证明。我们使用归纳法证明事实 2。对于 $l' = 0$，$d_0 = 0$ $l' = 1$ – 2 。对于，我们有 $d_1 = -\frac{1}{2}\xi_1$，因此事实显然成立。现在假设对于 $l' \le l$，事实 2 成立。那么，对于 $l' = l+1$，由事实 1 可得以下等式：

$$\begin{aligned}
d_{l+1} &= 2\mathcal{A}d_l - d_{l-1} - \xi_l \\
&= -\mathcal{A}U_{l-1}(\mathcal{A})\xi_1 - \sum_{j=2}^{l-1} 2\mathcal{A}U_{l-j}(\mathcal{A})\xi_j \\
&\quad + \frac{1}{2}U_{l-2}(\mathcal{A})\xi_1 + \sum_{j=2}^{l-2} U_{l-j-1}(\mathcal{A})\xi_j - \xi_l \\
&= -\frac{1}{2}U_l(\mathcal{A})\xi_1 - \sum_{j=2}^{l-2}(2\mathcal{A}U_{l-j}(\mathcal{A}) - U_{l-j-1}(\mathcal{A}))\xi_j - 2\mathcal{A}\xi_{l-1} - \xi_l \\
&= -\frac{1}{2}U_l(\mathcal{A})\xi_1 - \sum_{j=2}^{l-2} U_{l-j+1}(\mathcal{A})\xi_j - U_1(\mathcal{A})\xi_{l-1} - U_0(\mathcal{A})\xi_l \\
&= -\frac{1}{2}U_l(\mathcal{A})\xi_1 - \sum_{j=2}^{l} U_{l+1-j}(\mathcal{A})\xi_j
\end{aligned}$$

因此，事实 2 对 $l' = l+1$ 成立。事实 2 的证明至此完成。□

引理 A.10 由事实 2 立即推出。

回顾公式 10，我们将该项表示为以下形式：

$$\mathbf{v}_1^T p(\mathcal{A})\mathbf{v}_1 - \hat{\mathbf{v}}_1^T\hat{\mathbf{V}}p(\hat{\mathbf{T}})\mathbf{e}_1 = \sum_{l=0}^{k} c_l \mathbf{v}_1^T d_l$$

其中，$d_l = -\frac{1}{2}U_{l-1}(\mathrm{A})\xi_1 - \sum_{j=2}^{l-1} U_{l-j}(\mathrm{A})\xi_j$ 由引理 A.10 得出，而 $\xi_l = 2\mathbf{E}T_l(\hat{}\mathbf{T})\mathbf{e}_1$ 由定义得出。同样，根据定义，$\mathbf{E} = [\delta_1, \delta_2, ..., \delta_k]\hat{}$，其中 $\delta_i = (AMV(\mathrm{A}, \hat{}\mathbf{v}_i) - \mathrm{A}\,\hat{}\mathbf{v}_i) - \hat{\alpha}_i\hat{}\mathbf{v}_i|_{\mathcal{V}-S_i} - \beta_i\hat{}\mathbf{v}_{i-1}|_{\mathcal{V}-S_{i-1}}$ 对每个 $i \le k$ 都成立。在我们的算法 4 中，我们选择 $S_i = \{u \in \mathrm{V} : |\hat{}\mathbf{v}_i(u)| > \epsilon d_u\}$。因此，要得到 $d_l$ 的上界，我们应先给出 $\delta_i$ 的上界。

Lemma A.11. $|\delta_i(u)| \le 3\epsilon d_u$ *for all* $i \le k, u \in \mathcal{V}$.

证明。回顾算法 4 的第 4–8 行，我们只对满足 $|\hat{}\mathrm{v}_i(u)| > \epsilon\sqrt{d_u d_v}$ 的边 $(u,v) \in \mathrm{E}$ 执行局部矩阵-向量乘法 $AMV(\mathrm{A}, \hat{}\mathrm{v}_i)$，因此误差 $|AMV(\mathrm{A}, \hat{}\mathrm{v}_i)(u) - \mathrm{A}\,\hat{}\mathrm{v}_i(u)|$ 仅由满足 $|\hat{}\mathrm{v}_i(u)| \le \epsilon\sqrt{d_u d_v}$ 的边 $(u,v) \in \mathrm{E}$ 导致。此外，由于我们在每次迭代 $i \le k$ 中选择 $S_i = \{u \in \mathrm{V} : |\hat{}\mathrm{v}_i(u)| > \epsilon d_u\}$，因此对于所有 $i \le k, u \in \mathrm{V}$，均有 $|\hat{}\mathrm{v}_i|_{\mathcal{V}-S_i}(u)| \le \epsilon d_u$。因此，我们得到：

$$\begin{aligned}
|\delta_i(u)| &\le |AMV(\mathcal{A}, \hat{\mathbf{v}}_i)(u) - \mathcal{A}\hat{\mathbf{v}}_i(u)| \\
&\quad + |\hat{\alpha}_i\hat{\mathbf{v}}_i|_{\mathcal{V}-S_i}(u)| + |\hat{\beta}_i\hat{\mathbf{v}}_{i-1}|_{\mathcal{V}-S_{i-1}}(u)| \\
&\le \sum_{v \in \mathcal{N}(u)} \frac{\epsilon\sqrt{d_u d_v}}{\sqrt{d_u d_v}} + \epsilon d_u + \epsilon d_u \le 3\epsilon d_u
\end{aligned}$$

$\hat{}$ 第二个不等式成立，是因为根据假设 1，$\hat{}\alpha_l, \beta_l$ 由常数界定；为简洁起见，我们记 $\hat{}\alpha_l$、$\hat{}\beta_l \le 1$。由此得证引理A.11。□

接下来，我们使用引理A.11和定义 $\xi_l = 2\mathbf{E}T_l(\hat{}\mathbf{T})\mathbf{e}_1$，为 $\xi_l$ 给出上界。

Lemma A.12. $|\xi_l(u)| \le 6\sqrt{k}\epsilon d_u$ *for all* $l \le k, u \in \mathcal{V}$.

证明。根据定义，$\xi_l = 2\mathbf{E}T_l(\hat{}\mathbf{T})\mathbf{e}_1$ 且 $\mathbf{E} = [\delta_1, \delta_2, ..., \delta_k]$。定义 $c_l = T_l(\hat{}\mathbf{T})e_1$，则 $\|c_l\|_2 = \|T_l(\hat{}\mathbf{T})\mathbf{e}_1\|_2 \le \|\mathbf{e}_1\|_2 = 1$。$\hat{}$ 第一个不等式成立，是因为 $\|T_l(\mathbf{T})\|_2 \le 1\hat{}$，因为根据假设 1，$\lambda(\mathbf{T}) \subset [\lambda_{min}(\mathrm{A}) \; \lambda_2(\mathrm{A})] \subset [-1, 1]$；并且根据切比雪夫多项式的性质，$\max_{x \in [-1,1]} |T_l(x)| \le 1$。因此，由 $\xi_l = 2\mathbf{E}c_l$ 和 $|\delta_i(u)| \le 3\epsilon d_u$，可得：

$$\begin{aligned}
|\xi_l(u)| &= |\sum_{j=1}^{k} 2c_l(j)\delta_j(u)| \le 6\epsilon d_u \|c_l\|_1 \\
&\le 6\sqrt{k}\epsilon d_u \|c_l\|_2 \le 6\sqrt{k}\epsilon d_u
\end{aligned}$$

这就证明了引理 A 12 . □

接下来，我们利用引理 A. 12 $d_l = -\frac{1}{2}U_{l-1}$ $\xi_1$ – 以及引理 A. 中的公 $\sum_{j=2}^{l-1} U_{l-j}$ $\xi_j$ 10 式 (A) (A)，证明 $d_l(s)$ 和 $d_l(t)$ 的上界。然而，为此我们首先需要第二类切比雪夫多项式 $\{U_l(x)\}_{l \le k}$ 的上界。

Lemma A.13. $\|\mathbf{D}^{1/2}U_l(\mathbf{P})\mathbf{e}_u\|_1 \le Ck$ *for* $u = s, t, \forall l \le k$.

证明。我们利用一条说明 $\{T_l(x)\}$ 与 $\{U_l(x)\}$ 之间关系的性质来证明该引理：$U_l(x) = xU_{l-1}(x)+T_l(x)$。利用这一性质以及定义 $C_1 = \max_{u=s,\ t;l\le k} \|\mathbf{D}^{1/2}T_l(\mathbf{P})\mathbf{e}_u\|_1$，对于 $u = s,\ t$，我们有：

$$\begin{aligned}
\|\mathbf{D}^{1/2}U_l(\mathbf{P})\mathbf{e}_u\|_1 &= \|\mathbf{D}^{1/2}\mathbf{P}U_{l-1}(\mathbf{P})\mathbf{e}_u + \mathbf{D}^{1/2}T_l(\mathbf{P})\mathbf{e}_u\|_1 \\
&\le \|\mathbf{P}\|_1\|\mathbf{D}^{1/2}U_{l-1}(\mathbf{P})\mathbf{e}_u\|_1 + \|\mathbf{D}^{1/2}T_l(\mathbf{P})\mathbf{e}_u\|_1 \\
&= \|\mathbf{D}^{1/2}U_{l-1}(\mathbf{P})\mathbf{e}_u\|_1 + \|\mathbf{D}^{1/2}T_l(\mathbf{P})\mathbf{e}_u\|_1 \\
&\le \sum_{j=1}^{l}\|\mathbf{D}^{1/2}T_j(\mathbf{P})\mathbf{e}_u\|_1 \\
&\le C_1 l \le C_1 k
\end{aligned}$$

其中，第二个不等式是通过递归重复这一过程得到的。这就证明了引理 A.13。□

有了引理 A.13，我们现在可以证明 $d_l$ 的上界。

Lemma A.14. *$|d_l(u)| \le 6C_1k^{5/2}\epsilon\sqrt{d_u}$ for $u = s,t$ and $l \le k$.*

证明。根据引理 A.10，我们有 $d_l = -\frac{1}{2}U_{l-1}(\mathbf{A})\xi_1 - \sum_{j=2}^{l-1}U_{l-j}(\mathbf{A})\xi_j$。因此，

$$\begin{aligned}
|d_l(s)| &= \left|\frac{1}{2}U_{l-1}(\mathcal{A})\xi_1(s) + \sum_{j=2}^{l-1}U_{l-j}(\mathcal{A})\xi_j(s)\right| \\
&\le \sum_{j=1}^{l-1}\left|U_{l-j}(\mathcal{A})\xi_j(s)\right| \\
&\le \sum_{j=1}^{l-1}\sum_{u\in\mathcal{V}}\left|\mathbf{e}_s^T U_{l-j}(\mathcal{A})\mathbf{e}_u\right||\xi_j(u)| \\
&\le \sum_{j=1}^{l-1}\sum_{u\in\mathcal{V}}\frac{\sqrt{d_s}}{\sqrt{d_u}}\left|\mathbf{e}_u^T U_{l-j}(\mathbf{P})\mathbf{e}_s\right||\xi_j(u)| \\
&\le 6\sqrt{k}\epsilon\sum_{j=1}^{l-1}\sum_{u\in\mathcal{V}}\sqrt{d_s d_u}\left|\mathbf{e}_u^T U_{l-j}(\mathbf{P})\mathbf{e}_s\right| \\
&\le 6\sqrt{k}\epsilon\sum_{j=1}^{l-1}\sqrt{d_s}\|\mathbf{D}^{1/2}U_{l-j}(\mathbf{P})\mathbf{e}_s\|_1 \le 6C_1k^{5/2}\epsilon\sqrt{d_s}
\end{aligned}$$

其中，最后一个不等式由 $\|\mathbf{D}^{1/2}U_l(\mathbf{P})\mathbf{e}_s\|_1 \le C_1k$ 和 $l \le k$ $|d_l\ t\ | \le$ 得出。用同样的论 $6C_1k^{5/2}\epsilon\sqrt{d_t}$ 14 证，我们也可以证明 ( ) 。这就完成了引理 A. 的证明。□

Finally, by $\mathbf{v}_1^T p(\mathcal{A})\mathbf{v}_1 - \hat{\mathbf{v}}_1^T\hat{\mathbf{V}}p(\hat{\mathbf{T}})\mathbf{e}_1 = \sum_{l=0}^{k} c_l\mathbf{v}_1^T d_l$, and $\mathbf{v}_1 = \hat{\mathbf{v}}_1 = \frac{s}{\epsilon\sqrt{d_s}} - \epsilon\sqrt{\frac{t}{\sqrt{d_t}}/\frac{1}{d_s}} + \frac{1}{d_t}$，我们有如下不等式：

$$\begin{aligned}
|\mathbf{v}_1^T p(\mathcal{A})\mathbf{v}_1 - \hat{\mathbf{v}}_1^T\hat{\mathbf{V}}p(\hat{\mathbf{T}})\mathbf{e}_1| &= |\sum_{l=0}^{k} c_l\mathbf{v}_1^T d_l| \\
&\le \sum_{l=0}^{k} c_l\left|\frac{d_l(s)}{\sqrt{d_s}} - \frac{d_l(t)}{\sqrt{d_t}}\right|/\sqrt{\frac{1}{d_s}+\frac{1}{d_t}} \\
&\le 6C_1k^{5/2}\epsilon\sum_{l=0}^{k} c_l/\sqrt{\frac{1}{d_s}+\frac{1}{d_t}} \\
&\le 6C_1k^{5/2}\epsilon\sum_{l=0}^{k} 2c_l\sqrt{d} \\
&= 12C_1k^{5/2}\epsilon\left(\sum_{l=0}^{k} c_l\right)\sqrt{d}
\end{aligned}$$

其中 $d = \min\{d_s, d_t\}$由我们的定义给出。接下来，根据 $\sum_{l=0}^{k} c_l \le \kappa\log(\frac{\kappa}{\epsilon})$和 $\bar{\kappa} = \sqrt{\kappa}\log(\frac{\kappa}{\epsilon})$，$|\mathbf{v}_1^T p(\mathbf{A})\mathbf{v}_1 - \hat{\mathbf{v}}_1^T\hat{\mathbf{V}}p(\hat{\mathbf{T}})\mathbf{e}_1|$可被界定为：

$$\begin{aligned}
|\mathbf{v}_1^T p(\mathcal{A})\mathbf{v}_1 - \hat{\mathbf{v}}_1^T\hat{\mathbf{V}}p(\hat{\mathbf{T}})\mathbf{e}_1| &\le 12C_1\kappa^{2.25}\epsilon\sqrt{d}\log^{7/2}(\frac{\kappa}{\epsilon}) \\
&= \tilde{O}(C_1\kappa^{2.25}\epsilon\sqrt{d})
\end{aligned}$$

因此，通过设置 $\epsilon = \tilde{\Omega}(\frac{\epsilon\sqrt{d}}{\kappa^{2.25}C_1})$，我们可以使 $|\mathbf{v}_1^T p(\mathbf{A})\mathbf{v}_1 - \hat{\mathbf{v}}_1^T\hat{\mathbf{V}}p(\hat{\mathbf{T}})\mathbf{e}_1| \le \frac{\epsilon}{4}d$。最后，根据式 9，我们有：

$$\begin{aligned}
|\hat{r}_{\mathcal{G}}(s,t) - r_{\mathcal{G}}(s,t)|/(\frac{1}{d_s}+\frac{1}{d_t}) &\le |\mathbf{v}_1^T(\mathbf{I}-\mathcal{A})^{\dagger}\mathbf{v}_1 - \mathbf{v}_1^T p(\mathcal{A})\mathbf{v}_1| \\
&\quad + |\mathbf{v}_1^T p(\mathcal{A})\mathbf{v}_1 - \hat{\mathbf{v}}_1^T\hat{\mathbf{V}}p(\hat{\mathbf{T}})\mathbf{e}_1| \\
&\quad + |\hat{\mathbf{v}}_1^T\hat{\mathbf{V}}p(\hat{\mathbf{T}})\mathbf{e}_1 - \mathbf{e}_1^T(\mathbf{I}-\hat{\mathbf{T}})^{-1}\mathbf{e}_1| \\
&\le \frac{\epsilon'}{4} + \frac{\epsilon'}{4}d
\end{aligned}$$

等价地，

$$|\hat{r}_{\mathcal{G}}(s,t) - r_{\mathcal{G}}(s,t)| \le (\frac{\epsilon'}{4}+\frac{\epsilon'}{4}d)\frac{2}{d} \le \epsilon'$$

证明至此完成 . □

## A.7 定理 4.4 的证明。

定理 4.4（重述）算法 4 的期望运算次数有界于 $O(\frac{1}{\epsilon}\sum_{i\le k}(\|\hat{\mathbf{v}}_i\|_1 + \|\mathcal{A}\hat{\mathbf{v}}_i^+\|_1 + \|\mathcal{A}\hat{\mathbf{v}}_i^-\|_1) + k^{2.37})$。当设定 $k = \sqrt{\kappa}\log\frac{\kappa}{\epsilon}$，$\epsilon = \tilde{\Omega}(\frac{\epsilon\sqrt{d}}{\kappa^{2.25}C_1})$ 时，该结果为 $\tilde{O}(\kappa^{2.75}C_1C_2/(\epsilon'\sqrt{d}))$。

我们给出 Lanczos Push 时间复杂度的证明。回顾一下，在每次迭代 $i \le k$中，我们都会执行子集 Lanczos 递推：$\hat{\beta}_{i+1}\hat{\mathbf{v}}_{i+1} = AMV(\mathbf{A}, \hat{\mathbf{v}}_i) - \hat{\alpha}_i\hat{\mathbf{v}}_i|_{S_i} - \hat{\beta}_i\hat{\mathbf{v}}_{i-1}|_{S_{i-1}}$。因此，$i^{th}$ 步的时间复杂度分析分为两部分：(i) 界定计算 $AMV(\mathbf{A}, \hat{\mathbf{v}}_i)$ 所需的运算次数；(ii) 界定计算 $\hat{\alpha}_i,\beta_i$ 及其他加减运算所需的运算次数。

步骤 1。对于计算 $AMV(\mathbf{A}, \hat{\mathbf{v}}_i)$（算法 4 中第 4–8 行），回想一下，我们只在满足 $|\hat{\mathbf{v}}_i(u)| > \epsilon\sqrt{d_u d_v}$且 $\|\mathbf{A}\hat{\mathbf{v}}_i\|_1 \le C'$的边$(u,v)$上执行矩阵-向量乘法 $\mathbf{A}\hat{\mathbf{v}}_i$。在分析中，我们定义 $E_i = \{(u,v) \in \mathbf{E} : \hat{\mathbf{v}}_i(u) \ge \epsilon\sqrt{d_u d_v}\}$ 为计算 $AMV(\mathbf{A}, \hat{\mathbf{v}}_i)$ 时已搜索过的边的子集。此外，我们将$\hat{\mathbf{v}}_i$拆分为$\hat{\mathbf{v}}_i^+$和$\hat{\mathbf{v}}_i^-$，其中$\hat{\mathbf{v}}_i^+$和$\hat{\mathbf{v}}_i^-$分别表示$\hat{\mathbf{v}}_i$的正部和负部。类似地，我们定义 $E_i^+ = \{(u,v) \in \mathbf{E} : \hat{\mathbf{v}}_i^+(u) \ge \epsilon\sqrt{d_u d_v}\}$和 $E_i^- = \{(u,v) \in \mathbf{E} : \hat{\mathbf{v}}_i^-(u) \ge \epsilon\sqrt{d_u d_v}\}$。显然 $|E_i| = |E_i^+| + |E_i^-|$。接下来，我们分别界定 $E_i^+$和 $E_i^-$的大小。注意：

$$\begin{aligned}
\|\mathcal{A}\hat{\mathbf{v}}_i^+\|_1 &= \sum_{u\in\mathcal{V}}|\mathcal{A}\hat{\mathbf{v}}_i^+(u)| = \sum_{u\in\mathcal{V}}\left|\sum_{v\in\mathcal{N}(u)}\frac{\hat{\mathbf{v}}_i^+(u)}{\sqrt{d_u d_v}}\right| \\
&= \sum_{u\in\mathcal{V}}\sum_{v\in\mathcal{N}(u)}\frac{\hat{\mathbf{v}}_i^+(u)}{\sqrt{d_u d_v}} = 2\sum_{(u,v)\in\mathcal{E}}\frac{\hat{\mathbf{v}}_i^+(u)}{\sqrt{d_u d_v}} \\
&\ge 2\sum_{(u,v)\in E_i^+}\frac{\hat{\mathbf{v}}_i^+(u)}{\sqrt{d_u d_v}} \ge 2\epsilon|E_i^+|.
\end{aligned}$$

因此，$|E_i^+| \le O(\frac{\|\mathcal{A}\hat{\mathbf{v}}_i^+\|_1}{\epsilon})$。类似地，$|E_i^-| \le O(\frac{\|\mathcal{A}\hat{\mathbf{v}}_i^-\|_1}{\epsilon})$。因此，计算 $AMV$ 步骤 2 所需的总运算量为 $O(\frac{\|\hat{\mathbf{v}}_i^+\|_1+\|\hat{\mathbf{v}}_i^-\|_1}{\epsilon})$。

$|E_i| = |E_i^+| + |E_i^-| \le (A\hat{\mathbf{v}})$

我们注意到，其他运算的数量（算法 4 第 9–17 行）可由 $O(|S_i| + |S_{i-1}| + supp(\hat{\mathbf{v}}_{i+1}))$ 界定，因为我们只处理子集 $S_i, S_{i-1}$ 和 $supp(\hat{\mathbf{v}}_{i+1})$，其中 $supp(\hat{\mathbf{v}}_{i+1})$ 表示 $\hat{\mathbf{v}}_{i+1}$ 的非零项集合。我们设 $S_i = \{u \in V : |\hat{\mathbf{v}}_i(u)| > \epsilon d_u\}$，因此可以得到 $|S_i| \le \frac{\|\hat{\mathbf{v}}_\epsilon^i\|_1}{\epsilon}$ 的上界。同理，$|S_{i-1}| \le \frac{\|\hat{\mathbf{v}}_\epsilon^{i-1}\|_1}{\epsilon}$。此外，通过子集 Lanczos 迭代，我们注意到 $|E_i| + |S_i| + |S_{i-1}| \le 5 \cdot supp\{\hat{\mathbf{v}}_{i+1}\}$。

因此，算法 4 每次迭代 $i$ 的总运算量可由 $O(|E_i| + |S_i| + |S_{i-1}| + supp(\hat{\mathbf{v}}_{i+1})) = O(\frac{1}{\epsilon}(\|\hat{\mathbf{v}}_i^+\|_1 + \|\hat{\mathbf{v}}_i^-\|_1) + (\{\hat{\mathbf{v}}\}))$ 界定。因此，算法在总共进行迭代时的运算次数可由 $O(\frac{1}{\epsilon}\sum_{i\le k}(\|\hat{\mathbf{v}}_i\|_1 + \|A\hat{\mathbf{v}}_i^+\|_1 + \|A\hat{\mathbf{v}}_i^-\|_1))$ 界定。最后，计算最终输出 $\left(\frac{1}{d_s} + \frac{1}{d_t}\right)\hat{\mathbf{v}}_1^T\hat{\mathbf{V}}(\mathbf{I} - \hat{\mathbf{T}})^{-1}\mathbf{e}_1$ 需要 $O(k^{2.37})$ 时间。证毕。□

## A.8 推论 4.5 的证明

推论 A.16。（重述推论 *4.5*）算法 *4* 的运行时间为 $\tilde{O}(\kappa^{2.75}\sqrt{nm/(\epsilon\sqrt{d})})$，用于计算 $r_{\mathcal{G}}(s,t)$ 的 $\epsilon$-绝对误差近似。具体而言，$C_1 \le \sqrt{m}$ 和 $C_2 \le 3\sqrt{n}$。

首先，根据切比雪夫多项式的性质，对于任意 $i \in \mathbb{N}^+$，我们有 $\max_{x\in[-1,1]}|T_i(x)| = 1$。由于概率转移矩阵 $\mathbf{P}$ 的特征值位于 $[-1,1]$ 中，因此有 $\|T_i\|_2 \le 1$，$\|T_i(\mathbf{P})\mathbf{e}_u\|_2 \le 1$，$\|T_i(\mathbf{P})\mathbf{e}_u\|_1 = \|d_u^{1/2}\mathbf{x}_i(u)\| \le (\sum_{u\in V} d_u)^{1/2}(\sum_{u\in V}\mathbf{x}_i(u)^2)^{1/2} \le \sqrt{m}$。因此，$C_1 \le \sqrt{m}$。

在每次迭代中，我们维护 $\|\hat{\mathbf{v}}_i\|_2 = 1$。因此，再次利用柯西-施瓦茨不等式以及 $\|A\|_2 \le 1$，可得 $\|A\hat{\mathbf{v}}_i^+\|_1 \le \sqrt{n}\|A\hat{\mathbf{v}}_i^+\|_2 \le \sqrt{n}\|\hat{\mathbf{v}}_i^+\|_2 \le \sqrt{n}\|\hat{\mathbf{v}}_i\|_2 = \sqrt{n}$，并且对于 $\|A\hat{\mathbf{v}}_i^-\|_1$，同样的上界也成立。因此，$C_2 \le 3\sqrt{n}$。算法 4 的运行时间上界可通过 $\tilde{O}(\kappa^{2.75}\sqrt{mn/(\epsilon\sqrt{d})})$ 进行简化。

## A.9 引理 5.2 的证明

引理A.17。（引理 *5.2* 重述）如果 $\mathcal{G}_1$ 是按照第*5*节所述的方式构造的，则有 $\kappa(\mathcal{G}_1) = \Theta(n)$。

该证明受到 [2]中引理4.1的启发。回顾可知，条件数定义为 $\kappa = \frac{2}{\lambda_2}$，其中 $\lambda_2$ 是归一化拉普拉斯矩阵的第二小特征值。首先，根据切格尔不等式可得 $\lambda_2$ 的上界。根据导通率的定义，有 $\frac{2}{3n} = \phi(V(H_1)) \ge \phi_{\mathcal{G}_1} \ge \frac{1}{2}\lambda_2$，其中第一个等式成立，是因为Ramanujan图是3-正则图，并且$(s,t)$是 $V(H_1)$与 $V(H_2)$之间唯一的割边。因此，有 $\lambda_2 \le \frac{4}{3n} = O(\frac{1}{n})$。接下来，我们证明 $\lambda_2 \ge \Omega(\frac{1}{n})$。为此，我们考虑图$G_1$的拉普拉斯矩阵。根据构造，有 $L_{\mathcal{G}_1} = L_{H_1} + L_{H_2} + \delta_{s,t}\delta_{s,t}^T$，其中$\delta_{s,t}$是在 $s$取值1、在 $-1$ 处取值 $t$、其他位置取值0的指示向量。随后，对于任意向量 $x \perp \mathbf{1}$，我们将 $x$ 按$x = x_1 + x_2$分解，其中 $x_1$ 仅在 $u \in V(H_1)$中取值，$x_2$ 仅在 $u \in V(H_2)$中取值。由于 $x \perp \mathbf{1}$，我们定义$m = \frac{2}{n}\sum_{u\in V(H_1)} x_1(u) = -\frac{2}{n}\sum_{u\in V(H_2)} x_2(u)$。接下来，令$x_1' = x_1 - m\mathbb{I}_{V(H_1)}$，且 $x_2' = x_2 + m\mathbb{I}_{V(H_2)}$（$\mathbb{I}_{V(H_1}$是一个指示向量，在 $u \in V(H_1)$处值1，其他位置取值0）。现在

我们反设：对于任意小常数 $c$，都存在足够大的 $n$，使得 $\lambda_2(L_{\mathcal{G}_1}) \le \frac{c}{n}$。于是，存在某个$x \perp \mathbf{1}$满足$\|x\|_2 = 1$，使得 $x^T L_{\mathcal{G}_1} x \le \frac{c}{n}$。因此，我们可以得到如下方程。

$$x^T L_{\mathcal{G}_1} x = x_1^T L_{H_1} x_1 + x_2^T L_{H_2} x_2 + (x(s) - x(t))^2. \tag{11}$$

那么，上述等式右边的三项都 $\le \frac{c}{n}$。利用 $H_1$ 是扩展图这一事实（$L_{H_1}$ 的 $\lambda_2$ 可由 $H_1$ Ramanujan 图得到），我们有：

$$\frac{1}{2}\|x_1'\|_2^2 \le x_1'^T L_{H_1} x_1' = x_1^T L_{H_1} x_1 \le \frac{c}{n}. \tag{12}$$

因此，$\|x_1'\|_2 \le \sqrt{\frac{2c}{n}}$。类似地，$\|x_2'\|_2 \le \sqrt{\frac{2c}{n}}$。然而，由于 $\|x\|_2 = 1$，我们有：

$$1 = \|x\|_2 \le \|x_1'\|_2 + \|x_2'\|_2 + m\sqrt{n}. \tag{13}$$

因此，当设定 $c \le \frac{1}{32}$ 时，我们有 $m \ge \frac{1}{2\sqrt{n}}$。然而，另一方面，我们有以下不等式。

$$(x(s) - x(t))^2 = (x_1'(s) + m - (x_2'(t) - m))^2 \ge (2m - 2\sqrt{\frac{2c}{n}})^2 \ge \frac{1}{4n}. \tag{14}$$

这与 $(x(s) - x(t))^2 \le \frac{c}{n}$ 矛盾，因为 $c \le \frac{1}{32}$。因此，必定存在某个常数 $c$，使得对于任意充分大的 $n$，$\lambda_2(L_{\mathcal{G}_1}) \ge \frac{c}{n}$。因此，对于归一化拉普拉斯矩阵，我们有 $\lambda_2(\mathcal{L}_{\mathcal{G}_1}) \ge \frac{1}{4}\lambda_2(L_{\mathcal{G}_1}) = \Omega(\frac{1}{n})$。综合起来，我们有 $\lambda_2 = \lambda_2(\mathcal{L}_{\mathcal{G}_1}) = \Theta(\frac{1}{n})$。这等价于 $\kappa = \Theta(n)$。

## A.10 定理 5.3 的证明

定理 A.18.（定理 *5.3* 重述）任何以绝对误差 $\epsilon \le 0.01$ 近似 $r_{\mathcal{G}}(s,t)$、且成功概率为 $\ge 0.6$ 的算法，都需要 $\Omega(\kappa)$ 次查询。

将定理 5.1 与引理 5.2 结合，即可立即得到此结论。若存在一个算法，以绝对误差 $\epsilon \le 0.01$近似 $r_{\mathcal{G}}(s,t)$，则该算法可以区分 $G_1$ 和 $G_2$（因为根据定理 5.1，$G_1$ 和 $G_2$ 的 $s,t$-电阻距离值至少相差 0.01）。然而，根据定理 5.1，任何能够区分 $G_1$ 和 $G_2$ 的算法都需要 $\Omega(n)$ 次查询，而根据引理 5.2，$\kappa = \Theta(n)$。因此，任何需要 $\Omega(\kappa)$ 次查询的算法都能区分 $G_1$ 和 $G_2$。

## A.11 定理 6.2 的证明

定理 A.19.（重述定理 *6.2*）算法 *5* 计算 $\hat{\mathbf{f}}$，使得 $\|\mathbf{f} - \hat{\mathbf{f}}\|_1 \le \epsilon$，其中迭代次数设为 $k = O(\sqrt{\kappa}\log\frac{m}{\epsilon})$。

我们将 $\mathbf{\Pi} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^T$ 定义为 $\mathbf{L}$ 的像上的投影矩阵。根据势向量 $\boldsymbol{\phi}$ 的定义，采用与公式 3 相同的策略，可得以下公式：

$$\begin{aligned}
\boldsymbol{\phi} &= \mathbf{L}^\dagger(\mathbf{e}_s - \mathbf{e}_t) \\
&= \mathbf{\Pi}\mathbf{D}^{-1/2}(\mathbf{I} - \mathcal{A})^\dagger\mathbf{D}^{-1/2}(\mathbf{e}_s - \mathbf{e}_t) \\
&= \mathbf{\Pi}\mathbf{D}^{-1/2}(\mathbf{I} - \mathcal{A})^\dagger(\frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}}) \\
&\approx \sqrt{\frac{1}{d_s} + \frac{1}{d_t}}\mathbf{\Pi}\mathbf{D}^{-1/2}\mathbf{V}(\mathbf{I} - \mathbf{T})^{-1}\mathbf{V}\mathbf{v}_1 \\
&= \sqrt{\frac{1}{d_s} + \frac{1}{d_t}}\mathbf{\Pi}\mathbf{D}^{-1/2}\mathbf{V}(\mathbf{I} - \mathbf{T})^{-1}\mathbf{e}_1
\end{aligned} \tag{15}$$

根据定义 $\mathbf{f}(e) = \boldsymbol{\phi}(u) - \boldsymbol{\phi}(v) = (\mathbf{e}_u - \mathbf{e}_v)^T \mathbf{L}^{\dagger}(\mathbf{e}_s - \mathbf{e}_t)$，我们可以近似得到：

$$\begin{aligned}\hat{\mathbf{f}}(e) &= \sqrt{\frac{1}{d_s} + \frac{1}{d_t}}(\mathbf{e}_u - \mathbf{e}_v)^T \boldsymbol{\Pi}\mathbf{D}^{-1/2}\mathbf{V}(\mathbf{I} - \mathbf{T})^{-1}\mathbf{e}_1 \\ &= \sqrt{\frac{1}{d_s} + \frac{1}{d_t}}(\mathbf{e}_u - \mathbf{e}_v)^T \mathbf{D}^{-1/2}\mathbf{V}(\mathbf{I} - \mathbf{T})^{-1}\mathbf{e}_1\end{aligned} \tag{16}$$

这与算法 5 第 1—2 行中的公式等价。采用与定理 3.4 证明相同的方法，当设置迭代次数为 $k = O(\kappa \log \frac{\kappa m}{\epsilon}) = O(\kappa \log \frac{m}{\epsilon})$ 时，我们可以控制 $\hat{}$ 误差 $|\mathbf{f}(e) - \mathbf{f}(e)| \leq \epsilon/m$。因此，$\|\mathbf{f} - \hat{}\mathbf{f}\|_1 \leq \epsilon$。证明完毕。