在 $o(\log n)$ 轮次内实现大规模并行单源 SimRank

Siqiang Luo*南洋理工大学
新加坡
siqiang.luo@ntu.edu.sg

Zulun Zhu*南洋理工大学
新加坡
ZULUN001@e.ntu.edu.sg

arXiv:2304.04015v1 [cs.DC] 8 Apr 2023

摘要

SimRank 是一种最基础的度量，用于评估图中两个节点之间的结构相似性，并已应用于众多数据管理任务。这些任务通常涉及单源 SimRank 计算，用于评估源节点 s 与所有其他节点之间的 SimRank 值。由于计算复杂度高，对于大规模图而言，单源 SimRank 计算一直极具挑战性，因此近期研究转向分布式处理。令人惊讶的是，尽管 SimRank 已被广泛采用二十年，关于分布式 SimRank 的可证明结果理论研究却寥寥无几。

本文在大规模并行计算（MPC）模型下对单源 SimRank 计算进行理论研究。该模型是用于描述 MapReduce、Hadoop 或 Spark 等分布式系统的标准理论框架。现有分布式 SimRank 算法要么需要 $\Omega(\log n)$ 通信轮复杂度，要么需要 $\Omega(n)$ 机器空间来处理包含 n 个节点的图。我们突破了这一壁垒。具体而言，对于包含 n个节点的图，针对任意查询节点 v和常数误差 $\epsilon > \frac{3}{n}$，我们证明，机器之间只需进行 $O(\log^2 \log n)$ 轮通信，便几乎足以计算单源 SimRank 值，且绝对误差至多为 ϵ；与此同时，每台机器所需空间仅为 n的次线性规模。据我们所知，这是首个能够突破 $\Theta(\log n)$ 轮复杂度壁垒、且具有可证明结果准确性的 MPC 单源 SimRank 算法。

CCS 概念

• **计算理论 → 大规模并行算法、分布式算法**。

关键词

SimRank、分布式计算、通信轮次

ACM 参考格式：

Siqiang Luo 和 Zulun Zhu[1]。2023。 $o(\log n)$ 轮次中的大规模并行单源 Sim-排名。载于 *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (*会议简称 *'XX)*。ACM，纽约，NY，美国，14 页。https://doi.org/XXXXXXX.XXXXXXX

*作者按字母顺序排列。

允许免费制作本作品的全部或部分内容的数字版或纸质副本，供个人或课堂使用，但前提是不得为营利或商业利益制作或分发副本，且副本首页须注明本声明和完整引文。对于本作品中由 ACM 以外的其他方拥有版权的组件，必须尊重其版权。注明出处后允许摘录。其他复制或再版、发布到服务器或重新分发至列表，均须事先获得明确许可和/或支付费用。请向 per missions@acm.org 申请许可。会议简称 *'XX，2023*年*6*月*3*日至*5*日，纽约州伍德斯托克© 2023 美国计算机协会。ACM ISBN 978-1-4503-XXXX-X/18/06...$15.00 https://doi.org/XXXXXXX.XXXXXXX

1 引言

在图中评估两个节点之间的结构相似性，是许多数据管理和数据挖掘任务的基础。示例包括推荐系统 [1, 2], 避免客户流失 [3], 垃圾邮件检测 [4], 链接预测 [5] 和图挖掘 [6–9]。在众多相似性度量中，SimRank [10] 是图上应用最广泛的度量之一。SimRank基于这样的直觉定义：只有当两个节点的邻居节点相似时，这两个节点才相似。形式上，它使用以下递归方程计算两个节点 u和 v之间的 SimRank，其中 c是一个正的常数因子， $I(u)$表示节点 u的入邻居集合。

$$s(u,v) = \begin{cases} 1, & u = v \\ \frac{c}{|I(u)||I(v)|} \sum_{u' \in I(u)} \sum_{v' \in I(v)} s(u', v'), & u \neq v \end{cases} \quad (1)$$

自 Jeh 和 Widom 提出以来， [10], SimRank 在各种应用领域日益流行，例如社交分析 [11], 协同过滤 [12], 以及最近邻搜索 [13]。

1.1 分布式 SimRank 计算

我们专注于 单源*SimRank*计算，其重要性已在大量近期研究中得到揭示 [14–21]。给定一个图 G，针对源节点 s 的单源 SimRank 评估 s 与图中所有其他节点之间的 SimRank 值。单源 SimRank 计算广泛应用于需要根据某个对象对其他对象进行排名的场景。例如，它可以应用于搜索引擎，以定位与给定对象最相似的网页 [22],，或应用于社交网络服务，为给定用户推荐新朋友 [1, 23],，或作为排名度量来对对象进行聚类 [24]。

对于包含 n 个节点的图而言，计算单源 SimRank 对大规模图来说颇具挑战，因为这本质上需要进行 $O(n)$ 次成对 SimRank 计算，而每次计算的成本都可能过高。尤其是，按照公式 1 中的递归形式，计算 $s(u,v)$ 需要访问图中的许多节点对，因此复杂度达到 $O(n^2)$。为了解决效率问题，近期的研究 [25–27]采用基于随机游走的方法来近似 SimRank 值。其核心思想是将 SimRank 计算转化为估计两个源节点发出的两条衰减型随机游走的相遇概率（详情请参见第 2.2 节）。此时，计算复杂度取决于采样的随机游走数量，从而实现显著加速。然而，即使采用了改进后的方法，当 n 较大时，计算单源 SimRank 仍然颇具挑战。

因此，越来越多的人开始将分布式计算 [14, 15, 28]应用于 SimRank计算，这涉及多台机器以协作方式计算 SimRank 值，

最终将计算扩展到大规模图。现有的分布式 SimRank 算法（例如 [14, 15, 28]）大多侧重于实证评估，而具有可证明近似结果保证的非平凡理论分析则鲜有涉及。本文旨在对分布式环境中的单源 SimRank 值计算进行非平凡分析。我们关注近似 SimRank 算法，其输出的 SimRank 值的绝对误差至多为 ϵ，其中 $\epsilon \in (0,1)$ 是给定的误差阈值。我们基于一种著名的分布式计算模型——大规模并行计算（MPC） [29–32], ——研究这一课题。该模型已广泛用于分布式算法的理论分析，也应用于分析各种数据挖掘和图处理任务 [33–37]。MPC 考虑一组机器，每台机器可容纳 S 个字的空间。机器数量 M 设置为 $\Theta(F/S)$ 中的某个整数，以便分布式系统能够容纳输入，其中 F 表示输入的大小，$\tilde{\Theta}(\cdot)$ 相较于 $\Theta(\cdot)$ 隐藏了对数因子 [1]。在 MPC 中，每台机器持有部分数据，并通过同步的消息传递通信轮次与其他机器交换本地结果，同时受限于每台机器每轮发送或接收的消息大小至多为 S。

在对分布式计算进行建模时，每台机器上的本地计算成本通常远低于机器之间的同步成本。因此，按照惯例，在大规模并行计算（MPC）分析中会省略本地计算成本，重点放在减少通信轮次上。此外，通信轮次数量与每台机器的空间之间存在内在的权衡。考虑在一个包含 m 条边和 n 个节点的图上进行 SimRank 计算。$S = \Omega(m+n)$ 是一种简单情形，因为所有计算都可以在一台机器上本地完成，因此只需 $O(1)$ 轮即可。然而，对于大规模图，更重要的是考虑次线性空间设置（即 $S = o(n)$）；这种设置在许多分布式系统和现有研究中都很常见（例如 [33, 34, 46]）。在这种设置下，机器之间必须进行通信，才能获取整个图的视图；其目的是减少单源 SimRank 近似所需的通信轮次。

开放问题。我们分析了现有具有代表性的分布式 SimRank 算法，并在表12中总结了它们对通信轮次和机器空间的要求。简而言之，它们要么需要 $\Omega(n)$ 的每台机器空间（UniWalk [28]），要么需要 $\Omega(\log n)$ 个通信轮次（DISK [14] 和 CloudWalker [15]）。我们指出，对于自然问题而言，在大规模并行计算中突破 $\Theta(\log n)$ 的轮次复杂度壁垒通常颇具挑战，也因此受到广泛关注 [34, 36, 46–50]，以期将分布式计算的轮数降至 $O(\mathbf{poly}(\log\log n))$。在第3节中，我们还将说明，基于最先进的 SimRank 算法提出的直观想法为何无法实现这一目标。因此，一个自然的开放的理论问题随之提出：

给定误差阈值 $\epsilon \in (0,1)$，将输出 *SimRank* 的近似单源 *SimRank* 算法定义为一种算法具有 ϵ 绝对误差的值。是否存在一种分布式近似单源 *SimRank* 算法，能够在包含 n 个节点的图上于 $o(\log n)$ 轮次内完成，同时每台机器只需 $o(n)$ 空间？

1.2 我们的主要结果

在本文中，我们对上述开放问题给出了肯定的回答，并提出了一种适用于大规模并行计算（MPC）模型 [29]的分布式单源 SimRank 算法。具体而言，对于一个含有 n 个节点和 m 条边的图，我们研究的 MPC 模型包含一组机器，每台机器的空间为子线性空间 $S = n^{\alpha}$，其中 $\alpha \in$ （0，1）。我们的主要结果可由以下定理表述。

定理 1. 给定一个含有 n 个节点和 m 条边的图中的查询源节点 s，存在一种算法，能够使用 M台机器，在 $O\left(\log^2 \log n\right)$个通信轮次内，以高概率 [3]计算 s 与所有其他图节点之间具有 ϵ绝对误差保证的 *SimRank* 值。该算法仅要求每台机器的空间为 $S = n^{\alpha}$，其中 $\alpha < 1$，且 M是 $\tilde{O}((m+n)/S)$的某个取值。

据我们所知，这是首个在仅需每台机器的强次线性空间的情况下，实现亚对数 n 通信轮次的分布式单源 SimRank 算法。

2 预备知识

我们首先介绍所采用的分布式模型。然后讨论SimRank 计算的基本预备概念和算法。表 2 列出了常用的符号。

2.1 分布式计算模型

分布式计算是大数据时代应对各种应用 [14, 15, 28, 47, 51 56] –

的最重要技术之一。近年来，大规模并行计算（ MPC ） [29] 已成为用于刻画分布式算法复杂度的热门理论框架，因为它能够高度模拟一般分布式计算的情形。MPC 模型有三个重要参数：输入数据规模 F、所涉及的机器数量 M，以及每台机器上的空间容量（字） S。要求在给定 F 和 S 时，机器数量 M 应为 $\tilde{O}(\frac{F}{S})$，其中 $\tilde{O}(\cdot)$ 隐藏了一个多对数因子。我们对 MPC 模型的主要关注源于以下几个方面：

空间。 考虑一个包含 n 个节点的输入图。图算法的大规模并行计算可分为三种类型：强**超线性**空间（$S = n^{1+\omega}$），其中 $\omega > 0$为某个常数；近**线性** 空间（$S = \Theta(n)$）；以及强**次线性**空间（ $S = n^{\alpha}$ ），其中 $\alpha \in$ 为某个常数（ 0， 1 ）。超线性模型通常可以采用局部算法，但这会使其失去在大规模数据上应用的一般性。因此，许多研究聚焦于更能体现分布式系统可扩展性的次线性设置。本文关注强次线性设置，在此设置下，每台机器甚至无法存储全部图节点。

[1]这一设定与许多已有研究一致，例如 [38–41]。我们还注意到，MPC 领域的其他一些研究，例如 [42–45], , 规定了更严格的要求，即 $M = \Theta(F/S)$。[2]由于篇幅所限，相关工作的分析留待附录 A。由于作者没有基于大规模并行计算模型进行分析，我们自行对其进行了分析。

[3]我们称某事件以高概率发生，是指存在一个常数 $\tau > 0$，使得该事件发生的概率至少为 $1 - \frac{1}{n^{\tau}}$，其中 n 表示图节点的数量。 —

表1：包含 n 个节点和 m 条边的图上分布式单源 SimRank 计算的通信轮次。我们考察现有的分布式或并行 SimRank 算法，并从通信轮次和每台机器的空间两个方面进行分析。分析基于同步轮次通信。每轮总空间开销是各台机器所占空间之和。每台机器的空间指单台机器最坏情况下的空间开销。由于工作负载不均衡，每台机器的最坏情况空间复杂度可能与总空间复杂度相同。

算法	轮次	每轮总空间开销	精度误差	每台机器的空间开销
CloudWalker [15] [1]	$O(\frac{nl(I+l)}{b})$	$O\left(\frac{bl^2\log\frac{n}{\epsilon_p^2}}{\epsilon_p^2}+n+m\right)$	不提供保证	$O\left(\frac{bl^2\log\frac{n}{\epsilon_p^2}}{\epsilon_p^2}+\frac{m+n}{M}\right)$
UniWalk [28] [2]	$O(l)$	$O\left(n^2 l\log\frac{n}{\epsilon^2}\right)$	ϵ	$O\left(n^2 l\log\frac{n}{\epsilon^2}\right)$
DISK [14][3]	$O\left(\frac{\log^2 n}{\epsilon_d^2}+K\right)$	$O(m\log n+Kn)$	$\frac{c(1-c^K)}{1-c}\epsilon_d+c^{K+1}$	$O\left(\frac{m\log n+Kn}{M}\right)$
我们的[4]	$O\left(\log^2\log n\right)$	$O\left(m\log n+n^{1+o(1)}\log^{4.5}n+\frac{n\log^2}{(\epsilon-\frac{3}{n})^2}\right)$	ϵ	$O(n^\alpha)$

[1] ϵ_p 是估计随机游走分布时的误差，b 是控制单机处理节点数量的参数，I 是雅可比方法中的迭代次数。l 是用户定义的游走长度或步数。[2] l 是用户定义的游走长度或步数。[3] K 是线性化 SimRank 的截断项数量，ϵ_d 是内部估计误差阈值。[4] $\alpha<1$ 和 $n^\alpha>$ 对数[5] n/(对数 $\frac{1}{\sqrt{c}}$)。

表 2：常用符号。

符号说明	描述
$G(V,E)$	有向图 G，节点集 V 和边集 E
n,m	$n=\|V\|, m=\|E\|$
$s(v,u)$	节点 v 和 u 之间的 SimRank分数
$\tilde{s}(v,u)$	节点 v 和 u 之间的 SimRank分数估计值
l	随机游走的长度或步数
ϵ	SimRank 分数的误差阈值
c	SimRank 分数定义中的衰减因子
M	机器总数
S	每台机器的可用空间
α	用于衡量空间的常数因子
N_l	长度为 l 的随机游走的实际数量

通信轮次。 MPC 模型中的计算基于通信轮次进行。初始时，每条边都会被随机分配给一台机器。我们假设每个节点都有一个从 1 到 n的整数 ID，并且每台机器都有一个从 1 到 M的整数 ID。计算以同步轮次的方式进行。在一轮开始时，每台机器可能会接收到上一轮中其他某些机器发送的消息。在一轮中，每台机器根据其本地数据或接收到的消息执行本地计算。随后，每台机器会将计算结果打包成消息，发送给目标机器。每台机器都会创建消息包并将其路由到网络中，因此，在一轮中每台机器发送/接收的消息大小不得超过空间容量 S。只有在上一轮结束后，新的一轮才会开始。程序执行所需的轮次数称为轮复杂度 [57]，它描述了分布式计算的成本，因为主要成本通常来自机器之间代价高昂的通信。在 MPC 模型中设计算法的主要原则，是在每台机器的空间约束下实现较低的轮复杂度。

MPC 算法的存在性。 给定每台机器的空间 $S=n^\alpha$，如果对于任意 $n\geq n_0$（其中n_0为常数），都存在一种满足空间约束的分布式算法，并且该算法使用 M台机器和 $M=\tilde{O}(\frac{m+n}{S})$，则称 MPC 算法存在。为了便于分析 MPC 算法的存在性，我们首先给出一个关于机器空间扩展的准备引理如下。

引理 1. 如果存在一个 $O(f(n))$轮的分布式图算法，对于任意 $0<\alpha<1$，使用 $M=\tilde{O}(\frac{m+n}{n^\alpha})$ 台机器时，该算法的每台机器空间为 $S=\overline{\Theta(n^\alpha)}$，且 $f(n)$是一个与 α 无关的函数（即 α 对轮复杂度仅贡献一个常数因子），那么就存在一个每台机器空间为 $S=n^\alpha$、且轮复杂度相同的 *MPC* 算法。

该引理消除了机器空间按常数因子扩增时分析轮复杂度所面临的障碍。例如，假设我们有一个分布式算法，满足以下条件：（1）其轮复杂度中隐藏着 α 因子；（2）适用于任意 $\alpha\in(0,1)$，其中 $M=\tilde{O}(\frac{m+n}{n^\alpha})$；（3）每台机器的空间为 Cn^α，其中某个常数为 C。那么，引理 1 保证存在一个机器空间为 n^α 的 MPC 算法。

2.2 SimRank 近似计算

给定一个有向图 $G=(V,E)$，并令 $n=|V|$和 $m=|E|$。遵循 [16, 17],，我们的目标是计算常数误差下的近似 SimRank 值。具体而言，给定任意源节点 $s\in G$和一个常数误差 ϵ，我们旨在计算 s与任意其他节点 $u\in G$之间的 SimRank 值，使得 $|s(s,u)-\tilde{s}(s,u)|\leq\epsilon$，其中 $s(s,u)$ 表示真实的 SimRank 值，而 $\tilde{s}(s,u)$ 表示估计值。

根据公式 1 迭代计算 SimRank 值可能会占用大量内存空间，并产生高昂的计算成本。因此，当前最先进的方法采用了以下 $\sqrt{c}$ 衰减游走计算，该方法最初由 [20]提出。

引理 2。 对于任意两个节点 u、$v \in G$，u与v之间的 *SimRank* 等于从 u和 v出发、在 $\bar{G}$上进行的两个 $\sqrt{c}$ 衰减游走的相遇概率；如果存在一个整数 $i \geq 0$，使得两个游走在第 i步访问同一个节点，则称这两个游走相遇。

这里，从节点 u 在 $\bar{G}$ 上的 $\sqrt{c}$-衰减游走是一种遍历 $\bar{G}$的过程：在游走的每一步，它都有 $1-\sqrt{c}$ 的概率停留在当前节点，否则就跳转到当前节点的一个随机选取的出邻居。这里，我们将长度为l的游走定义为包含 $l+1$ 个节点和 l条边的路径。我们也称长度为l的游走有 l步。特别地，从给定的起始节点生成长度为i的游走的概率为$(\sqrt{c})^i \cdot (1-\sqrt{c})$。

蒙特卡罗方法。 通过从 u和 v中采样 N对 $\sqrt{c}$-游走，如果其中 H对游走相遇，那么 $\frac{H}{N}$就是$s(u, v)$的估计值。采样数量~~N~~控制估计精度。

3 预热：一种在 $O(\log n)$ 轮内完成、使用伪线性空间的算法

本节介绍一种算法，它将用于单源 SimRank 近似的基于 $\sqrt{c}$-游走的计算直接迁移到大规模并行计算模型中。我们证明，通过适当设置参数，该算法能以高概率（即概率至少为 $1-\frac{1}{n}$）在 $O(\log n)$ 轮内完成计算，同时所需的单机空间为伪线性空间 $S = \tilde{O}(n)$。

该算法的主要思想是改造基于 $\sqrt{c}$-游走的方法，并通过设置样本量 $N = \log_{2\epsilon^2} \frac{2n^2}{} = O(\log \frac{n}{\epsilon^2})$ 来应用蒙特卡罗方法；也就是说，从给定的源节点 u以及反向图 $\bar{G}$中的任意其他节点 v出发，采样 $\sqrt{c}$-游走。根据蒙特卡罗方法，我们需要从 u和 v生成 N对随机游走，并确定其中有多少对游走相遇。游走相遇的概率 $\frac{H}{N}$可用于估计 $s(u, v)$。~~一~~设随机变量 $X_i = 1$ 在第 i对中的两条游走相遇时取值为 $X_i = 1$，否则取值为 $X_i = 0$。根据 Hoeffding 不等式 [58]，我们有以下不等式：

$$\begin{aligned}\mathbf{Pr}\left[\left|s(u,v) - \frac{H}{N}\right| \geq \epsilon\right] &= \mathbf{Pr}\left[\left|\sum_{1\leq i\leq N} \mathbb{E}(X_i) - \sum_{1\leq i\leq N} X_i\right| \geq \epsilon\right] \\ &\leq 2e^{-2N\epsilon^2} \text{ (By Hoeffding's Inequality)} \\ &= 1/n^2 \end{aligned} \tag{2}$$

由于共有 n 个 SimRank 估计，根据并集界，所有 SimRank 估计的误差至多为 ϵ，其概率至少为 $1-\frac{1}{n^2} \cdot n = 1-\frac{1}{n}$。

接下来，我们证明，可以并行地从每个节点分别采样 N 条随机游走，并且在 $O(1)$ 轮内并发地让所有游走各前进一步，同时机器空间为 $S = \tilde{O}(n)$。具体而言，我们为每条边 (u, v) 分配一个 ID $u \cdot n + v$。在大规模并行计算中，我们可以用 $O(1)$ 个通信轮次按 ID 对 [59, 60] 这些边排序，并按顺序将它们从机器 1 存储到机器 M。为了分隔连续节点的邻居集合，我们为每个节点 v插入一条特殊边占位符 $(v, 0)$，其 ID 为 $v \cdot n$，并将其与实际边一同排序。排序后，占位符 $(v, 0)$ 会被放在以 v开头的第一条边之前。通过排序，每条边也知道自己在排序中的排名。于是，节点 v有 $r_{v+1} - r_v - 1$ 个邻居，其中 r_v（分别地，r_{v+1}）表示 $(v, 0)$（分别地，$v+1$）的排名。计算 $\{r_{v+1} - r_v - 1 | v \in V\}$ 可以分两轮完成：~~假~~设每台机器有 $S = O(n)$，那么可以将所有 $\{r_v | v \in V\}$ 发送到同一台机器，以计算 $\{r_{v+1} - r_v - 1 | v \in V\}$，再将 $r_{v+1} - r_v - 1$ 发送回持有边 $(v, 0)$ 的机器。这样不会~~破~~坏 $O(n)$ 空间，因为集合 $\{r_{v+1} - r_v - 1 | v \in V\}$ 的大小为 $O(n)$。此外，为了存储随机游走，当以节点 u为终点的游走路径 P沿着边 (u, v) 从节点 u延伸到 v时，路径 $P \cup (u, v)$ 将被发送到持有 $(v, 0)$ 的机器。

由于随机游走以 $\sqrt{c}$ 衰减，~~随~~机游走长度至少为 $3\log_{1/\sqrt{c}} \frac{n}{\epsilon}$ 的概率至多为 ϵ^3/n^3。因此，高概率下，所有 Nn 个随机游走的长度均为 $O(\log \frac{n}{\epsilon})$。注意到计算长度为 l 的随机游走需要 $O(l)$ 轮，因此，高概率下，计算所有随机游走的轮复杂度为 $O(\log \frac{n}{\epsilon})$。此外，由于随机游走共有$Nn = O(\frac{n}{\log_{\epsilon^2} n})$ 个，且高概率~~下~~每次游走的长度为 $O(\log \frac{n}{\epsilon})$，因此，每轮机器之间发送的消息数至多为 $O(\log n \cdot Nn) = O(\frac{n}{\log n} \frac{n}{\epsilon^2 \log}) = \tilde{O}(n)$，且该结论以高概率成立（~~假设~~ ~~$\epsilon$ 为常数~~）。

最后，要计算 $\{s(u, v) | v \in V\}$，还需要并行检查有多少对游走相遇，其中每一对包含一条来自 u 的游走和另一条来自 v 的游走。正如我们将在第 5.1 节中很快解释的那样，这一步也可以在 $O(1)$ 轮次内完成。

总之，我们提出的算法能够在大规模并行计算中，以高概率在 $O(\log \frac{n}{\epsilon})$ 轮次内完成近似 SimRank 计算，所需单机空间为 $S = O(\frac{n}{\log n} \frac{n}{\epsilon^2 \log})$。~~可以看~~出，直接改造现有的基于随机游走的算法，~~很~~难突破$O(\log n)$轮次的瓶颈，因为在这类设计中，每一步随机游走都需要 $O(1)$ 轮次，而随机游走以高概率包含 $O(\log n)$ 步。同时，进一步减少单机空间也颇具挑战，因为可能存在枢纽节点，被来自不同节点的许多随机游走经过。

4 $O(\text{对数}^2 \text{对数}\, n)$-轮算法，使用次线性空间

在本节中，我们展示了基于上一节所介绍算法的若干重要改进，最终将轮复杂度降低至 $O(\log^2 \log n)$，并将每台机器的空间降至强次线性。

为了更好地理解我们的设计，我们概述了直接改造基于 $\sqrt{c}$-游走的方法所面临的挑战。首先，$\sqrt{c}$-衰减游走的最大长度可能是无限的；在分布式环境中，直接计算长度为 l 的游走，很容易需要 l 个通信轮次，因为当某些邻居位于不同机器上时，一步操作可能就需要一轮通信。在第3节中，我们给出了分析，表明通信轮次高概率下可被限制在 $O(\log \frac{n}{\epsilon})$。这类技术的一个缺点是，轮次数无法确定性地限定。其次，要计算节点 s 的单源 SimRank 值，需要从 s 和所有其他节点运行随机游走，因此会进行大量随机游走。这可能导致每台机器的空间开销很高，因为某些枢纽节点容易被许多随机游走经过

，这也是第3节中的算法无法实现严格次线性每台机器空间的主要原因。因此，需要仔细设计大规模并行计算算法，以确保轮次数较少且每台机器的空间开销较低。第三，在大规模并行计算模型中计算出随机游走后，它们会存储在不同的机器上。因此，检测两条游走是否相遇可能会使机器的空间开销过载。

4.1 基于游走的方法的新解读

为应对上述挑战，我们使用以下三种更适合大规模并行计算 $\bar{}$ 的操作，以批量方式重新诠释节点 u 和节点 v 之间基于 $\sqrt{c}$-游走的 SimRank 计算：

(a)随机游走生成。我们不生成长度非确定性的 $\sqrt{c}$-衰减随机游走，而是生成具有确定性长度分布的随机游走。特别地，为了从 u生成 N $\sqrt{c}$-游走，我们将根据游走长度的相应几何分布，为 $i \geq 0$生成 $N \cdot \sqrt{c}^i(1-\sqrt{c})$ 个长度为-i的游走。对于从 v开始的随机游走，也执行类似操作。固定随机游走长度更加适合大规模并行计算。正如我们将在第5.1节中展示的那样，我们可以仅用 $O\left(\log^2 \log n\right)$ 个大规模并行计算轮次，就从源节点 s 和每个节点 $v \in V$ 生成足够多的此类随机游走。

(b) 随机游走洗牌与匹配。 由于从 s（或 v）出发的随机游走是按长度递增的顺序生成的（这是步骤 (a) 导致的），因此直接将从 s 和 v 出发的随机游走配对进行相遇检测，无法有效估计 SimRank 值，因为从 s 和 v 出发的第 i 个游走彼此相关（它们的长度相同）。为保持随机性，我们将步骤 (a) 中计算得到的随机游走打乱。

(c) SimRank 计算。 最后，我们通过检测有多少对游走相遇，并根据引理 2 估计 SimRank 值 $s(s, v)$。

4.2 主要算法步骤概述

基于新的解释，我们在算法 1 中给出主要算法步骤，其中包含以下五个主要阶段。为便于说明，本节先概述主要思路，并将与大规模并行计算相关的详细操作留待第 5 节介绍。

初始状态。 SimRank 算法基于输入图的拓扑信息。初始时，应按照大规模并行计算模型的假设，将整个图划分到不同机器上。我们假设每台机器持有图中边的一个随机划分（算法 1 的第 1 行），因此每台机器大约持有 $\frac{m}{M}$ 条边。我们还会反转这些边，使图表示 $\bar{G}$（算法 1 的第 2 行）。

并行随机游走生成。 此阶段对应于前文提到的操作 (a)。由于我们需要计算从任意源节点 s 到所有其他节点的单源 SimRank 值，因此我们从每个节点生成 N $\sqrt{c}$ 条随机游走。此外，我们将它们截断至 $\log_{1/\sqrt{c}} n$ 长度（算法 1 的第 3 行）。通过设置 $N = \log 2\ \epsilon{-}3\ n(/)^2$'，我们表明，截断随机游走对最终精度的影响可以忽略不计（见第 5 节），并且我们可以保证

算法 1： 总体算法

输入： 图 $G=(V,\ E)$；源节点 s；衰减因子 c；蒙特卡洛失败概率 δ；精度误差 ϵ

输出： 每个 $u \in V$ 的 SimRank 分数 $\tilde{s}(s,\ u)$

1 将 G 存储在多台机器上；
2 在每台机器上反转 G 的边，以形成 $\bar{G}$；
3 $l \leftarrow \log_{\frac{1}{\sqrt{c}}} n$；
4 **对于** $i = 0$ **到** l 并行**执行**
5 $N_i = \left\lceil \frac{2n}{\epsilon^2} \cdot (\sqrt{c})^i (1-\sqrt{c}) \right\rceil$；
6 **对每个** 节点 $v \in V$ 并行 **执行**
7 从节点 v 生成 N_i 随机游走，在 $O(\log^2 \log n)$ 内轮次；
8 打乱并分解随机游走（参见第 5.2 节和第 5.3 节）；
9 运行**算法 2**，为每个 $u \in V$ 计算 $\tilde{s}(s,\ u)$；

SimRank 值的 ϵ误差界。另一个重要问题是，在生成 $N_i = N \cdot (\sqrt{c})^i$ 条长度为 i 的随机游走时，$\bar{N_i}$ 可能不是整数。在第 6 节中，我们给出一种舍入技术，并证明舍入后仍能保证误差界。我们还证明，这些随机游走可在 MPC 模型中通过$O(\log^2 \log n)$ 个通信轮次生成（算法 1 的第 4–7 行，详见第 5.1 节）。

洗牌随机游走。 为了保证蒙特卡洛模拟中每对随机游走的随机性，生成的随机游走必须进行洗牌（算法 1 的第 8 行）。MPC 模型中的洗牌操作不像单机环境中那样简单，因为来自同一节点的生成游走可能位于不同的机器上。对这些游走进行洗牌会产生机器间通信，这可能违反每台机器的空间限制。我们将在第 5 节详细讨论这一问题。

分解随机游走。 洗牌后，我们分别将源节点 s 出发的第 j 个游走与其他节点出发的第 j 个游走配对，并检测每对游走是否相遇。在单机环境中，检测两条游走是否相遇相对简单，因为所有信息都可以在本地加载。然而，在分布式环境中，当游走存储在不同机器上时，我们需要多次检测配对后的两条游走是否相遇。这带来了截然不同的挑战。为应对这些挑战，我们将生成的随机游走分解为元组，每个元组包含游走中一个被访问的节点（起始节点除外）。每个元组包含五个元素，用于传达检测游走是否在指定节点与另一条游走相交所需的全部信息。如图 1 所示，一条游走 $\{s,\ v_1,\ v_2,\ v_3\}$ 被分解为三个元组 $(j,\ 1,\ v_1,\ 3,\ s)$、$(j,\ 2,\ v_2,\ 3,\ s)$、$(j,\ 3,\ v_3,\ 3,\ s)$，其中 j 表示该元组由从 s 出发的游走中的第 j 个游走分解而来。元组中的其余元素依次为游走步数、特定游走步数处访问的节点、游走中的总步数和源节点。

计算SimRank 值。 评估 s 和 u 之间的SimRank 值，基于从 s 和 u 出发的两次游走相遇的概率。这可以通过对分解元组进行排序来检测（算法 $1\rightarrow$ 的第9 行，算法2的第 1行）。图1 分别展示了从 s 和 u 出发的第 j 次游走。这两次游走

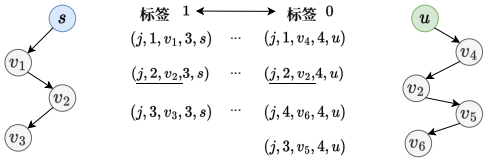

图 1：使用元组关联检测会面。

因为它们在第三个游走步中共享同一个节点 v_2 而相遇。对应于 v_2，来自两次游走的两个分解元组分别记为 $(j, 2, v_2, 3, s)$和 $(j, 2, v_2, 4, u)$。我们注意到它们共享前三个元素。一般来说，容易验证，如果两次游走相遇，那么必然存在两个分解元组，分别来自其中一次游走，并且它们共享元组中的前三个元素。更正式地，假设有两个从 v_0和 u_0出发的 j 次游走 $\{v_0, v_1, \ldots, v_{l_1}\}$和 $\{u_0, u_1, \ldots, u_{l_2}\}$ $(0 \le l_1, l_2 \le \log 1/\sqrt{c}\, n)$。游走$\{v_0, v_1, \ldots, v_{l_1}\}$ 被分解为 l_1个元组，记为 (j, i, v_i, l_1, v_0) $(1 \le i \le l_1)$；游走$\{u_0, u_1, \ldots, u_{l_2}\}$ 被分解为 l_2 个元组，记为 (j, k, u_k, l_2, u_0) $(1 \le k \le l_2)$。那么，如果两次游走相遇，或者等价地说，它们在相同步骤 i中共享同一个节点 v_i，则必然有 $v_i = u_i$，并且对应的两个元组（对于 v_i和 u_i）共享前三个元素。同样，挑战在于如何在机器之间适当地重新排列元组，以及如何在每台机器内收集和聚合游走相遇情况。我们将在第 5 节介绍MPC细节。

通过上述五个阶段，我们强调了我们的方法与经典的基于 $\sqrt{c}$- 游走的算法之间的主要差异：**(a)** 游走的长度被规范化为几何分布，并且较长的游走会被截断；我们在第 6节中证明，这不会违反误差容限；**(b)** 大多数步骤都经过了非平凡的重新设计，从而使所有步骤都能在 MPC 模型中高效实现（详见第 5 节）。

5 大规模并行计算操作详解

我们从基于 MPC 模型的实现角度，更详细地介绍第 4.2 节中的算法。

5.1 并行随机游走生成

在 MPC 模型中执行随机游走会产生通信轮次，因为每个节点下一个采样到的邻居可能位于不同的机器上。考虑节点 v_0 在游走中采样其邻居节点 v_1 的情况。在 MPC 模型中，由于 v_0 和 v_1 可能位于不同的机器上，生成长度为 $v_0 \to v_1$ 的游走可能需要一个通信轮次。一般而言，在 MPC 模型中生成一个长度为 l 步的随机游走通常需要 l 个通信轮次。为了减少通信轮次，我们首先介绍 WRME [46], 提出的一个结果，然后在定理 3 中给出我们的扩展：

定理 2。令 G 为一个有向图。令 N 和 l 为正整数，使得 $l = o(S)/\log^3 n$，其中 S为每台机器的可用空间。对于从 G中的每个节点 v开始，采样 N条长度为 l的独立随机游走这一任务，存在一种 *MPC* 算法，可在 $O\left(\log^2 \log n + \log^2 l\right)$轮次内运行，并使用$O\left(m + n^{1+o(1)} l^{3.5} + Nnl^{2+o(1)}\right)$总空间，且具有强次线性

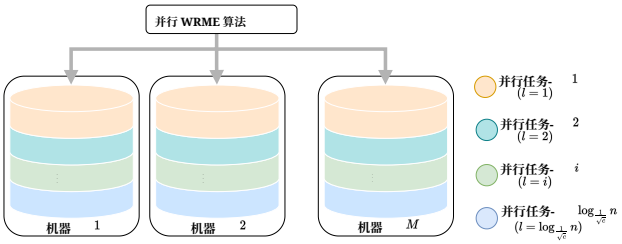

图 2：每台机器划分其空间，以并行处理多个任务。

每台机器的空间 $S = n^{\alpha}$ $(0 < \alpha < 1)$。该算法是不完美采样器，失败概率为 $1 - O\left(n^{-1}\right)$。

本质上，定理2 表明，在 MPC 模型中，从每个节点生成长度为-l的随机游走可以高效地完成轮次。特别地，只需经过 $O\left(\log^2 \log n + \log^2 l\right)$通信轮次，即可生成 N个长度为-l的随机游走，且每个游走具有不同的源节点。然而，即使有定理2，如果采用原始的 $\sqrt{c}$-衰减游走方法，我们仍然无法直接为 SimRank评估给出合理的轮次复杂度。原因在于，$\sqrt{c}$-衰减游走的长度 l 可能是无限的。为了解决这一问题，正如我们在算法 1第3 行和第 5行中所展示的，我们仔细设计了截断游走长度（第 3行）以及针对每个游走长度所要采样的随机游走数量（第 5 行）。当 $l = \log_{1/\sqrt{c}} n$ 时，截断长度保证了 $O\left(\log^2 \log n\right)$通信轮次的合理上界，而随机游走样本数量则确保了 SimRank 估计精度（我们将在第 6 节中对此进行形式化证明）。

并行随机游走 $O(\log^2 \log n)$轮次。为有效应用定理 2，我们令任务-i 负责从每个节点生成长度为 N_i 的-i 游走。我们让所有任务-i $(1 \le i \le \log_{1/\sqrt{c}} n)$ 在大规模并行计算中并行执行。我们在 MPC 模型中并发应用 $\log_{1/\sqrt{c}} n$ WRME算- 法的 [46] 个实例并发地，其中每个实例对应一个任务。

遗憾的是，并发执行多个 WRME 算法会导致每台机器的空间开销高于 n^{α}，因为默认情况下每个算法实例的本地开销最高可达 n^{α}。为解决这一问题，我们令 β 满足 $n^{\beta} = n^{\alpha}/\log_{1/\sqrt{c}} n$，并针对每个 Task-$i$使用 $S = n^{\beta}$并发应用 WRME 算法。具体而言，每台机器的空间 n^{α} 被划分为大小为 n^{β} 的子空间，用于计算Task-i，如图2所示。通过对轮复杂度和总空间开销进行仔细分析，我们给出以下定理。

定理 3。 算法 *1* 中的随机游走生成可以在 $O\left(\log^2 \log n\right)$轮次内以高概率完成，机器空间为 $S = n^{\alpha}$使得 $n^{\alpha} > \log^5 n/(\log \frac{1}{\sqrt{c}})$，总空间为$O\left(m \log n + n^{1+o(1)} \log^{4.5} n + \frac{n}{\log^2 (\epsilon^{-\frac{3}{n}})^2}\right)$。

我们注意到，WRME算法是一个不完美采样器，其失败概率至多为 $O(n^{-1})$。我们仅应用采样器 $O(\log n)$次，这仍然可以轻松保证我们的算- 法以高概率成功（因为对于任意常数 $\tau > 0$，$O(\log n) = o(n^{\tau})$）。

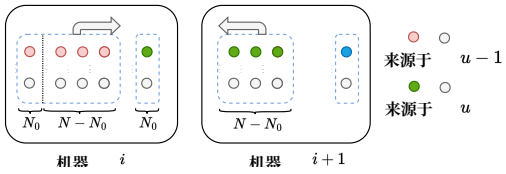

图3：交换 $N-N_0$ 条随机游走。5.2 随机游走洗牌

我们需要对从每个节点出发生成的 N 随机游走进行洗牌，以完全模拟生成 $\sqrt{c}$-游走的行为。进行洗牌的原因是，当我们选取从两个节点出发的两条随机游走并计算它们的相遇概率时，这两条游走的长度不一定相同。这一洗牌操作有助于我们保证所生成的 $\sqrt{c}$-衰减游走的蒙特卡洛模拟中的随机性。

在 O轮（1）内洗牌。 我们注意到，在大规模并行计算中，只要元素数量为 $O(n^\eta)$（其中 $\eta > 0$ 为某个常数），排序就可以在 O 轮（1） [60] 内完成。注意到 $N = O(\log n)$，因此存在 η，使得 $nN = O(n^\eta)$。因此，在大规模并行计算中，将所有 nN 条游走放在一起排序，可以在常数轮内完成；两条游走的比较依据是它们各自游走中的对应节点之间的比较。这样一来，所有从同一节点出发的游走都会聚集在一起。

对于足够大的 n，我们有 $n^\alpha > 2N\log_{1/\sqrt{c}} n$，而从同一节点出发、占用 $N\log_{1/\sqrt{c}} n$ 空间的游走可以存放在单台机器上。由于所有游走都已排序，并且 $N \frac{n^\alpha}{\log_{1/\sqrt{c}} n}$ 可能不是整数，因此从同一节点出发的每组 N 条游走可能会分布在两台机器上。设 W_u 为从节点 u 出发的游走集合。我们讨论两种对同一源节点的游走进行洗牌的情况。

情况1： 如果 W_u 中的游走完全位于一台机器上，则在本地进行洗牌。我们对所有此类游走集合进行洗牌，并在洗牌后为 W_u 中的每条游走分配从1到 N 的编号。

情况2： 如果源自同一节点的游走位于两台连续的机器上。不失一般性地，我们将源节点记为 u，第一台机器中有 N_0 个游走，第二台机器中有 $N-N_0$ 个游走。然后，我们将第二台机器中的这 $N-N_0$ 个游走与源自节点 $u-1$ 的最后 $N-N_0$ 个游走交换，如图3所示。由于对于足够大的 n，$n^\alpha > 2N\log_{1/\sqrt{c}} n$，因此可以保证那些源自 $u-1$ 的 $N-N_0$ 个游走位于第一台机器中，并且已在情况1中进行本地混洗。交换后，W_u 内的混洗可以在本地完成。我们注意到，通过保证 $n^\alpha > 2N\log_{1/\sqrt{c}} n$，可以确保交换不会发生冲突，从而能够在1轮中完成。

任何集合 W_u 都可以在情况1或情况2中进行打乱。打乱过程中，可以依次给每条游走标上从 1 到 N 的标签，第 j 条游走指的是被分配编号 j 的游走。

5.3 分解随机游走

分解所需 $O(1)$ 轮次。 回想一下，每条长度为-t的游走 $\{v_0, v_1, \ldots, v_t\}$ 都要分解成 t个元组，每个元组包含 5 个元素：(j, i, v_i, t, v_0)，其中 $1 \le i \le t$。元组 (j, i, v_i, t, v_0) 表示从 v_0出发的游走中，第 j 次随机游走的， the 游走的第 i 步访问节点 v_i，且该游走总共有 t 步。在 MPC 模型中，随机游走的分解可在持有该游走的机器内完成。尽管每条游走会生成多个元组，但总空间仅会扩大一个常数因子（即 5 倍），因为每个元组都对应游走中访问的一个节点。因此，此操作不会引发机器间的通信，且只会使空间开销以常数倍扩大；根据引理 1，这种影响可以忽略不计。我们用 D_G 表示所有分解元组的集合。

5.4 检测相遇游走

由于分解元组存储在不同的机器上，如果我们使用这些元组来检测某些随机游走对是否相遇，就会面临挑战。算法 2 展示了检测游走相遇并计算 SimRank 值的伪代码。

在 $O(1)$ 轮次内排序。 我们的第一步是对所有机器上的元组进行排序（第 1 行）。由于元组包含五个元素，因此采用多维排序：首先根据第一个元素排序，然后对第一个元素相同的元素根据第二个元素排序，依此类推。元素按照其节点 ID 排序。我们还特意让源节点 s 的节点 ID**小于**其他节点，以确保共享前三个元素的游走以源自 s的游走开头。

回顾第 4 节，我们证明了：如果两条游走相遇，那么必然存在两个分解元组，分别来自这两条游走，并且共享前三个元素。对这些元组进行排序后，共享前三个元素的元组将被聚集在一起；如果存在第五个元素为 s 的元组，它将被放置在该簇的第一个位置。具体而言，假设元组 (j, i, v, l_1, s)、(j, i, v, l_2, u_1)、(j, i, v, l_3, u_2) 共享前三个元素，这意味着从 s 开始的第 j 条游走，与从 u_1 和 u_2 开始的第 j 条游走之间存在相遇次数。排序后，这三个元组将按以下顺序聚集：(j, i, v, l_1, s)、(j, i, v, l_2, u_1)、(j, i, v, l_3, u_2)。正式结果如下。

引理 3。 如果从查询源节点s开始的第 j 个游走W与从 u开始的第 j 个游走 W'相遇次数，则必定存在一个由 W 分解而成的元组 T，以及一个由W'分解而成的元组 T'，二者的前三个元素相同。此外，在对这些元组进行排序后，T是由 W分解而成且排在 T'之前的最接近元组。

在 $O(1)$ 轮中进行游走配对。要计算 s与 u_1之间的 SimRank，我们需要将 (j, i, v, l_1, s) 和 (j, i, v, l_2, u_1) 配对；类似地，我们还需要将 (j, i, v, l_1, s) 和 (j, i, v, l_3, u_2) 配对，以计算 s与 u_2之间的 SimRank。配对可能颇具挑战，因为这些元组存储在不同的机器上，因此机器间通信不可避免。在大规模并行计算模型中，与其他机器通信的机器需要占用与通信消息大小相同的空间。为了合理限制机器间的通信成本，我们采用以下 PREDECESSOR 过程 [36]，将共享相同前三个元素的元组配对；该过程可在 $O(1)$ 个通信轮次内完成。

算法 2： 检测相遇并计算 SimRank

输入： 第5.3节中的元组集 D_G；源节点 s

输出： 每个 $u \in V$ 的估计 SimRank分数 $\tilde{s}(s, u)$

对 D_G 中的所有元素进行排序;

计算 $z = \frac{\log 2}{(\epsilon - \frac{3}{n})^2}$, $|Z_{u,l}| = z \cdot (\sqrt{c})^l \cdot (1 - \sqrt{c})$ 和 $|\hat{Z}_{u,l}| = \lceil |Z_{u,l}| \rceil$ 对于 $l = 1, \ldots, \log_{\sqrt{c}} n$ 和 $u \in V$;

对于 所有 元组 $(i, j, v_j, l, v) \in D_G$ 并行 **执行**

  如果 $v = s$ **那么**

    将 (i, j, v_j, l, v) 标记为 1

  else

    将 (i, j, v_j, l, v) 标记为 0

对 所有元组 (i, j, v_j, l_2, u) 并行标记为 *0* 的，**执行**

  使用以下方式将其连接到标记为 1 的最近元组 (i, k, v_k, l_1, s):

    前驱;

  如果 $(j = k) \wedge (v_j = v_k)$ **则**

    $\tilde{s}(s, u) \leftarrow \tilde{s}(s, u) + \frac{|Z_{s l_1}| |Z_{u l_2}|}{|Z_{s l_1}| |Z_{u l_2}|} \cdot \frac{1}{z}$

前驱：给定一个按序排列的元组列表，其中每个元组都标记为 *0* 或 *1*。对于每个标记为 *0* 的元组T，*PREDECESSOR* 会将其与满足以下条件的最近一个标记为 *1* 的元组 T关联：T在排序中位于 T，之前。对于任意常数 $\alpha > 0$，*PREDECESSOR* 都可以通过 n^α 每台机器的空间，在 $O(1)$ 个大规模并行计算轮次内实现。

为了应用前驱来配对元组，每台机器都可以运行局部算法，以区分从节点 s开始的随机游走所生成的元组与其他元组。如图1 所示，所有机器并行地将第五个元素为 s的元组标记为 1。否则，将元组标记为0。根据引理 3，我们可以应用前驱过程，将最近的标签-1元组（即从 s出发的游走所生成的元组）与其后续的每个标签-0 元组关联起来。这里，关联 意味着，持有标签-0 元组的机器能够知晓其之前最近的标签-1 元组。对于每一对这样的关联元组 T和 T，，如果它们的前三个元素相同，则 T，和 T对应的游走相遇。

5.5 计算 SimRank 值

上一步中配对的元组将对相应的 SimRank 值作出贡献。例如，假设来自 s和 u 的两条游走相遇，那么这对游走将对 SimRank 值 $s(s, u)$ 作出贡献，因为 $s(s, u)$是通过来自节点 s 和节点 u的游走的相遇概率来估计的。挑战在于如何在多台机器之间并发分发和聚合这些 *SimRank* 贡献 。

至关重要的是，让对应于同一对节点（例如，(s, u)）的 *SimRank* 贡献由同一台机器处理，以避免重复统计游走相遇次数。例如，游走 $\{s, u_1, u_2, w\}$ 和 $\{v, u_1, u_2, k\}$ 在节点 u_1和节点 u_2处均相遇。为确保精度，每次游走相遇事件都只能统计一次。为此，我们让每台机器专门处理相对于 s的 $\frac{n}{M}$ 个节点的 SimRank 评估。一般性而言，我们假设节点 ID 从 1 到 n。然后，第 i台机器处理节点 ID 属于 $\left[{}^{n}_{(i-1)M} + 1, \frac{ni}{M} \right]$的节点集。 ——— — 在 $O(1)$轮次中计算 SimRank。计算SimRank 的详细大规模并行计算操作如下所示（算法 2 第 8–11 行也进行了说明）。使用 PREDECESSOR过程，对于每个标签为 0 的元组（i、 j、 v_j、 l_2、 u），都可以将其链接到最近的元组（i、 k、 v_k、 l_1、 s）；如果 $j = k$ 和$v_j = v_k$，则表示两个元组所代表的两次游走相遇。随后，将发送一条游走相遇消息（j、 s、 l_1、 u、 l_2）（由持有元组（j、 i、 v、 l_1、 s）的机器发送）到负责计算SimRank s（s、 u）的目标机器。此处，游走相遇消息（j、 s、 l_1、 u、 l_2）表示：一条源自节点 s、长度为l_1的游走与一条源自节点 u、长度为 l_2 的游走相遇。此外，假设 s、 u 和 v是从 1 到 n 的节点 ID，则消息发送到的目标机器是第 $\lfloor \frac{uM}{n} \rfloor$ 台机器，因为该机器将计算 s（s、 u）。随后，每台机器可能会接收到多份游走相遇消息（j、 s、 l_1、 u、 l_2），但只保留其中一份，因为每个游走相遇事件只能计数一次。对于接收到的每条唯一消息（j、 s、 l_1、 u、 l_2），我们将值 $\frac{|Z_{s l_1}| |Z_{u l_2}|}{|Z_{s l_1}| |Z_{u l_2}|} \cdot \frac{1}{z}$加到 SimRank 值 $\tilde{s}(s, u)$ 中，其中 $z = \log_2 \left(\epsilon - \frac{3}{n}\right)^2$ 是游走样本的期望数量，$Z_{s,l}$ 表示源自 s 的长度为 l的实际生成随机游走集合，而$Z_{s,l}$是源自 s 的、预期生成的长度为 l 的随机游走集合。

我们区分实际样本和期望样本，因为$Z_{s,l}$可能不是整数。特别地，我们注意到 $|\hat{Z}_{s,l}| = \left\lceil z \cdot (\sqrt{c})^l \cdot (1 - \sqrt{c}) \right\rceil$，以及 $|Z_{s,l}| = z \cdot (\sqrt{c})^l \cdot (1 - \sqrt{c})$。当 $z \cdot (\sqrt{c})^l \cdot (1 - \sqrt{c})$ 不是整数时，期望游走集合$Z_{s,l}$基于大小为$\lceil |Z_{s,l}| \rceil$ 的集合 Z_1和大小为 $\lfloor |Z_{s,l}| \rfloor$ 的集合 Z_2 定义，使得以 $|Z_{s,l}| - \lfloor |Z_{s,l}| \rfloor$ 的概率从集合 Z_1 中选择一个 $Z_{s,l}$ 的游走，否则从集合 Z_2中选择。

游走相遇消息的空间分析。 每台机器收到的游走相遇消息数量，至多是该机器处理的节点所发起的随机游走数量的 $\log_{1/\sqrt{c}} n$ 倍。为说明这一点，在最坏情况下，对于由该机器处理的 u 生成的每个游走W的节点，都存在一个来自 s 的游走W，在该节点与 W 相遇，从而产生一条游走相遇消息。由于每台机器负责计算 n/M 个节点的 SimRank 值，因此每台机器用于接收消息的空间为

$$O\left(\frac{n}{M} \log_{\frac{1}{\sqrt{c}}} n \cdot \frac{\log 2n}{2(\epsilon - \frac{3}{n})^2}\right) = O\left(\frac{n \log^2 n}{M(\epsilon - \frac{3}{n})^2}\right)$$

因此，游走相遇消息的总空间开销为

$$O\left(\frac{n \log^2 n}{M(\epsilon - \frac{3}{n})^2} \cdot M\right) = O\left(\frac{n \log^2 n}{(\epsilon - \frac{3}{n})^2}\right)$$

6 结果总结

轮复杂度。轮复杂度主要由并行随机游走的生成过程决定，根据定理 3，其为 $O(\log^2 \log n)$。

空间复杂度 .游走相遇消息的总空间开销为 由随机游走生成的空间开销O主导 $\left(m \log n + n^{1+o(1)} \log^{4.5} n\right.$

（参见定理 3）。我们注意到，总空间开销为 $\tilde{O}(m+n)$，这表明对于 $S = n^{\alpha}$，存在 $M = \tilde{\Theta}(\frac{m+n}{S})$。

所有这些结果得出了定理 4，它也支持定理 1。

定理 4。 给定一个包含 n 个节点和 m 条边的图，以及一个常数误差 $\epsilon > 0$，对于足够大的 n，可以在大规模并行计算中运行算法 *1*，所需通信轮次为 $O\left(\log^2_{1+o(1)} n\right)$，每轮总空间为 $O\left(m \log n + n^{1+o(1)} \log^{4.5} n + \frac{n \log 2}{(\epsilon - \frac{3}{n})^2}\right)$，所需的每台机器的空间为 $S = n^{\alpha}$，使得 $n^{\alpha} > \log^5 n / (\log \frac{1}{\sqrt{c}})$。算法 *1* 是一个不完美采样器，其失败概率至多为 $O\left(n^{-1}\right)$。

精度分析。我们的随机游走采样技术的精度保证与简单的蒙特卡洛方法类似，但我们还采用了长度截断和游走次数舍入技术。这些技术使证明更加复杂，因此我们将证明留在附录中，仅列出主要结果如下。请注意，结果中的阈值 $\frac{3}{n}$ 还可以进一步降低（见附录）。

定理 5. 算法 *1* 输出 *SimRank* 值 $\tilde{s}(s,u)$，误差至多为 ϵ $(\epsilon \geq \frac{3}{n})$，且概率至少为 $1 - \frac{1}{n}$。

参考文献

[1] Phuong Nguyen、Paolo Tomeo、Tommaso Di Noia 和 Eugenio Di Sciascio。评估 SimRank 和个性化 PageRank，以构建数据网推荐系统。载于第 *24* 届万维网国际会议论文集，第 1477–1482 页，2015 年。[2] Dingheng Mo 和 Siqiang Luo。Agenda：演化图中的鲁棒个性化 PageRank。载于第 *30* 届 *ACM* 国际信息与知识管理会议论文集，第 1315–1324 页，2021 年。[3] Siqiang Luo、Xiaokui Xiao、Wenqing Lin 和 Ben Kao。高效的批量单跳个性化 PageRank。载于*2019* 年 *IEEE* 第 *35* 届国际数据工程会议（*ICDE*），第 1562–1565 页，2019 年。[4] András A Benczúr、Károly Csalogány 和 Tamás Sarlós。基于链接的相似性搜索，以打击网络垃圾信息。载于*AIRWEB*。Citeseer，2006 年。[5] David Liben-Nowell 和 Jon Kleinberg。社交网络中的链接预测问题。美国信息科学与技术学会期刊，58(7)：1019–1031，2007 年。[6] Ming Chen、Zhewei Wei、Bolin Ding、Yaliang Li、Ye Yuan、Xiaoyong Du 和 Ji-Rong Wen。通过双向传播实现可扩展的图神经网络。载于神经信息处理系统进展，33：14556–14566，2020 年。[7] Ruoming Jin、Victor E. Lee 和 Hui Hong。网络角色相似性的公理化排名。载于*KDD*，第 922–930 页。ACM，2011 年。[8] Ningyi Liao、Dingheng Mo、Siqiang Luo、Xiang Li 和 Pengcheng Yin。SCARA：通过面向特征的优化实现可扩展的图神经网络。*Proc. VLDB Endow.*，15(11)：3240–3248，2022 年。[9] Aleksandar Bojchevski、Johannes Klicpera、Bryan Perozzi、Amol Kapoor、Martin Blais、Benedek Rózemberczki、Michal Lukasik 和 Stephan Günnemann。利用近似 PageRank 扩展图神经网络。载于第 *26* 届 *ACM SIGKDD* 知识发现与数据挖掘国际会议论文集，第 2464–2473 页，2020 年。[10] Glen Jeh 和 Jennifer Widom。SimRank：一种结构上下文相似性度量。载于 *KDD*，第 538–543 页。ACM，2002 年。[11] Weiguo Zheng、Lei Zou、Yansong Feng、Lei Chen 和 Dongyan Zhao。面向大规模图的高效 SimRank 相似性连接。*Proc. VLDB Endow.*，6 504，2013 7 493( ): – 。[12] Ioannis Antonellis、Hector Garcia-Molina 和 Chi-Chao Chang。SimRank++：通过点击图链接分析进行查询重写。*Proc. VLDB Endow.*，1(1)：408–421，2008 年。[13] Pei Lee、Laks V. S. Lakshmanan 和 Jeffrey Xu Yu。前 k 个结构相似性搜索。载于 Anastasios Kementsietsidis 和 Marcos Antonio Vaz Salles（编），*2012* 年 *IEEE* 第 *28* 届国际数据工程会议（*ICDE 2012*），美国华盛顿特区（弗吉尼亚州阿灵顿），*2012* 年 *4* 月 *1–5* 日，第 774–785 页。IEEE计算机学会，2012 年。

[14] Yue Wang、Ruiqi Xu、Zonghao Feng、Yulin Che、Lei Chen、Qiong Luo 和 Rui Mao。DISK：一种具有精度保证的单源 SimRank 分布式框架。VLDB会刊，14(3):351–363，2020。[15] Zhenguo Li、Yixiang Fang、Qin Liu、Jiefeng Cheng、Reynold Cheng 和 John C. S. Lui。云中漫步：大规模并行 SimRank。VLDB会刊，9(1):24–35，2015。[16] Hanzhi Wang、Zhewei Wei、Ye Yuan、Xiaoyong Du 和 Ji-Rong Wen。大规模图上的精确单源 SimRank 计算。载于《2020年ACM SIGMOD数据管理国际会议论文集》，第653–663页，2020年。[17] Jieming Shi、Tianyuan Jin、Renchi Yang、Xiaokui Xiao 和 Yin Yang。Web规模图上的实时无索引单源 SimRank 处理。arXiv预印本 arXiv:2002.08082，2020。[18] Hanzhi Wang、Zhewei Wei、Yu Liu、Ye Yuan、Xiaoyong Du 和 Ji-Rong Wen。ExactSim：基于高精度基准真值的单源 SimRank 算法基准测试。VLDB期刊，30(6):989–1015，2021。[19] Takanori Maehara、Mitsuru Kusumoto 和 Ken-ichi Kawarabayashi。通过线性化高效计算 SimRank（本文发表事宜待询）。载于KDD，第1426–1435页。ACM，2014。[20] Boyu Tian 和 Xiaokui Xiao。Sling：用于 SimRank 的近最优索引结构。载于《2016年数据管理国际会议论文集》，第1859–1874页，2016。[21] Mitsuru Kusumoto、Takanori Maehara 和 Ken-ichi Kawarabayashi。用于 SimRank 的可扩展相似性搜索。载于《2014年ACM SIGMOD数据管理国际会议论文集》，第325–336页，2014。[22] Dániel Fogaras 和 Balázs Rácz。扩展基于链接的相似性搜索。载于《第14届万维网国际会议论文集》，第650–641页，2005年。[23] Guoming He、Haijun Feng、Cuiping Li 和 Hong Chen。通过迭代聚合在大规模图上并行计算 SimRank。载于KDD，第543–552页。ACM，2010。[24] Yuanzhe Cai、Pei Li、Hongyan Liu、Jun He 和 Xiaoyong Du。S-SimRank：结合内容和链接信息，高效地对论文进行聚类。载于《高级数据挖掘与应用国际会议》，第317–329页。Springer，2008。[25] Minhao Jiang、Ada Wai-Chee Fu 和 Raymond Chi-Wing Wong。READS：一种高效、准确的动态 SimRank 随机游走方法。VLDB会刊，10(9):937–948，2017。[26] Zhipeng Zhang、Yingxia Shao、Bin Cui 和 Ce Zhang。基于 SimRank 的相似性搜索算法的实验评估。VLDB会刊，10(5):601–612，2017。[27] Yingxia Shao、Bin Cui、Lei Chen、Mingming Liu 和 Xing Xie。面向大型动态图上 SimRank 的高效相似性搜索框架。VLDB会刊，8(8):838–849，2015。[28] Junshuai Song、Xiongcai Luo、Jun Gao、Chang Zhou、Hu Wei 和 Jeffrey Xu Yu。UniWalk：基于单向随机游走的大规模图可扩展 SimRank 计算。IEEE知识与数据工程汇刊，30(5):992–1006，2018。[29] Howard Karloff、Siddharth Suri 和 Sergei Vassilvitskii。MapReduce计算模型。载于《第二十一届年度ACM-SIAM离散算法研讨会论文集》，第938–948页。SIAM，2010。[30] Alexandr Andoni、Aleksandar Nikolov、Krzysztof Onak 和 Grigory Yaroslavtsev。几何图问题的并行算法。载于《第四十六届年度ACM计算理论研讨会论文集》，第574–583页，2014。[31] Paul Beame、Paraschos Koutris 和 Dan Suciu。并行查询处理的通信轮次。ACM学报，64(6):1–58，2017。[32] Michael T Goodrich、Nodari Sitchinava 和 Qin Zhang。MapReduce框架中的排序、搜索与模拟。载于《算法与计算国际研讨会》，第374–383页，2011。[33] Soheil Behnezhad、Sebastian Brandt、Mahsa Derakhshan、Manuela Fischer、MohammadTaghi Hajiaghayi、Richard M. Karp 和 Jara Uitto。稀疏图上的匹配与极大独立集的大规模并行计算。载于PODC，第481–490页。ACM，2019。[34] Artur Czumaj、Peter Davies 和 Merav Parter。通过低空间图稀疏化消除大规模并行计算中的随机性。ACM算法汇刊，17(2):16:1–16:27，2021。[35] Mohsen Ghaffari、Themis Gouleakis、Christian Konrad、Slobodan Mitrovic 和 Ronitt Rubinfeld。用于极大独立集、匹配和顶点覆盖的改进型大规模并行计算算法。载于PODC，第129–138页。ACM，2018。[36] Soheil Behnezhad、Laxman Dhulipala、Hossein Esfandiari、Jakub Lacki 和 Vahab S. Mirrokni。近最优的大规模并行图连通性算法。载于FOCS，第1615–1636页。IEEE计算机学会，2019。[37] Krzysztof Nowicki 和 Krzysztof Onak。大规模并行计算模型中支持批量更新的动态图算法。载于《2021年ACM-SIAM离散算法研讨会（SODA）论文集》，第2939–2958页。SIAM，2021。[38] Yi-Jun Chang、Manuela Fischer、Mohsen Ghaffari、Jara Uitto 和 Yufan Zheng。拥塞团模型、大规模并行计算和集中式局部计算中的 $(\delta+1)$ 着色复杂度。载于《2019年ACM

分布式计算原理研讨会论文集》，第471–480页，2019年。[39] Mohsen Ghaffari、Christoph Grunau 和 Slobodan Mitrović。b-匹配的大规模并行算法。载于《第34届ACM算法与体系结构并行性研讨会论文集》，第35–44页，2022年。[40] Amartya Shankha Biswas、Michal Dory、Mohsen Ghaffari、Slobodan Mitrović 和 Yasamin Nazari。距离近似与伸展子图的大规模并行算法。载于《第33届ACM算法与体系结构并行性研讨会论文集》，第118–128页，2021年。[41] Mohsen Ghaffari、Christoph Grunau 和 Ce Jin。树及更广泛图上的极大独立集、匹配和着色的改进MPC算法。arXiv预印本 arXiv:2002 09610，2020年。.[42] Miao Qiao 和 Yufei Tao。双属性无偏斜、孤立CP定理与大规模并行连接。载于Leonid Libkin、Reinhard Pichler 和 Paolo Guagliardo（编），PODS'21：第40届ACM SIGMOD-SIGACT-SIGAI数据库系统原理研讨会论文集，线上会议，中国，2021年6月20日至25日，第166–180页。ACM，2021。[43] Yufei Tao。基于规范边覆盖的并行无环连接。arXiv预印本 arXiv:2201.03832，2022。[44] Xiao Hu。覆盖或打包：大规模并行连接的新上界与下界。载于《第40届ACM SIGMOD-SIGACT-SIGAI数据库系统原理研讨会论文集》，第181–198页，2021。[45] Xiao Hu 和 易科。大规模并行连接算法。ACM SIGMOD记录，49(3):6–17，2020。[46] Jakub Lacki、Slobodan Mitrovic、Krzysztof Onak 和 Piotr Sankowski。随机、大规模且高效地行走。载于STOC，第364–377页。ACM，2020。[47] Siqiang Luo、Xiaowei Wu 和 Ben Kao。改进轮次复杂度的分布式PageRank计算。信息科学，607:109–125，2022。[48] Siqiang Luo。分布式PageRank计算：改进的理论研究。载于第三十三届AAAI人工智能会议（AAAI 2019），第4496–4503页。AAAI Press，2019。[49] Sebastian Brandt、Manuela Fischer 和 Jara Uitto。低内存MPC模型中均匀稀疏图上的匹配与极大独立集。CoRR，abs/1807 05374，2018年。.[50] Siqiang Luo。分布式PageRank计算中通信成本的改进——一项理论研究。载于《国际机器学习会议》，第6459–6467页。PMLR，2020。[51] Fabian Kuhn、Nancy Lynch 和 Rotem Oshman。动态网络中的分布式计算。载于《第四十二届ACM计算理论研讨会论文集》，第513–522页，2010。[52] Siqiang Luo、朱子辰、Xiaokui Xiao、Yin Yang、李春波 和 Ben Kao。顶点中心图系统中的多任务处理：评估与洞见。载于EDBT，第247–259页，2023。[53] Donald Kossmann。分布式查询处理的最新进展。ACM计算调查，32(4):422–469，2000。[54] Siqiang Luo、罗一峰、周水庚、丛高、管继宏 和 郑勇。道路网络上的分布式空间关键词查询。载于EDBT，第235–246页。Citeseer，2014。[55] Siqiang Luo、罗一峰、周水庚、丛高 和 管继宏。Disks：道路网络上分布式空间群组关键词搜索系统。VLDB会刊，5(12):1966–1969，2012。[56] Hartmut Klauck、Danupon Nanongkai、Gopal Pandurangan 和 Peter Robinson。大规模图问题的分布式计算。载于《第二十六届年度ACM-SIAM离散算法研讨会论文集》，第391–410页。SIAM，2014。[57] Mohsen Ghaffari、Fabian Kuhn 和 Jara Uitto。基于分布式下界的大规模并行计算条件困难性结果。载于FOCS，第1650–1663页。IEEE计算机学会，2019。[58] Wassily Hoeffding。有界随机变量之和的概率不等式。载于《Wassily Hoeffding文集》，第409–426页。Springer，1994。[59] Mohsen Ghaffari。大规模并行算法。2019。[60] Michael T. Goodrich、Nodari Sitchinava 和 Qin Zhang。MapReduce框架中的排序、搜索与模拟。载于ISAAC，计算机科学讲义第7074卷，第374–383页。Springer，2011。

一种缺失的复杂度分析

在表1中，我们梳理了相关文献，并给出了有关空间开销和轮复杂度的分析结果。我们注意到，这三种算法都旨在并行采样随机游走，以计算端点分布（CloudWalker）或两条路径的相遇概率（UniWalk 和 DISK）。按照 UniWalk 中的现有分析 [28], , 我们假设在分布式系统中执行一步随机游走需要一个通信轮次。

A.1 CloudWalker [15]

CloudWalker 使用以下定义来计算 SimRank:

$$S = cP^\top SP + D = \sum_{t=0}^{\infty} c^t P^{\top t} D P^t, \tag{3}$$

其中， S 是 SimRank矩阵， $S(i,\ j)$是第 i个节点与第 j个节点之间的SimRank分数， P是 G^T的转移矩阵，而 G^T是输入图 G 的转置， D 是 CloudWalker 旨在首先计算的对角修正矩阵。CloudWalker 的方法主要包括计算 D 和计算单源 $SimRank$。

计算 D。CloudWalker 依赖一种结合雅可比方法和蒙特卡罗方法的方法来计算对角修正矩阵 D ，其中每一行对应一个节点对其他节点的修正值。在假设输入图无法装入单机 ($S = n^\alpha$, $0 < \alpha < 1$) 的情况下，CloudWalker 无法一次处理所有节点，而是采用分布式结构来存储每个节点的数据。具体而言，每台机器一次将处理 b 个节点，并将 b 分配给一个待处理的节点编号。因此，CloudWalker 需要执行其算法 $\frac{n}{b}$ 次才能处理所有节点。我们注意到，在估计 D的过程中，CloudWalker 需要为每个节点采样 $N = (\log n)/\epsilon_p^2$ 长度为-l 的随机游走，其中 ϵ_p 是控制精度的参数。在采样 N 随机游走时，由于我们假设机器空间 $S < n$，每台机器可能需要与其他机器通信，以访问可能存储在其他机器中的邻接节点。例如，假设 $u \to v$ 是 E 中的一条边，且 u, v 存储在两台不同的机器中，则一个游走步 $u \to v$需要一轮通信。因此，采样一个长度为-l 的随机游走需要 $O(l)$轮通信。在运行 I次迭代、执行n/b 次之后，CloudWalker 的总体通信轮次为 $O(nIl/b)$。

然后，在雅可比方法中，CloudWalker需要 $O(blN)$的空间，在每台机器中存储游走状态矩阵 A。此外，还需要存储所有节点和边，产生 $O(m+n)$的空间开销。综上，每轮的空间开销为 $O\left((bl\log n)/\epsilon_p^2 + n + m\right)$。此外，每台机器需要 $O\left((bl\log n)/\epsilon_p^2\right)$ 的空间来存储 A。假设图在各台机器之间均匀分布，则每台机器的空间为 $O\left((bl\log n)/\epsilon_p^2 + (m+n)/M\right)$。

单源 SimRank 的计算。在 [15],CloudWalker 的算法 4 中，基于已获得的对角修正矩阵 D 计算单源 SimRank。在第 4–13 行中，CloudWalker 再次利用随机游走计算与 SimRank 结果对应的向量，这需要消耗 $O(l^2)$ 个通信轮次。通信轮次消耗的原因

也在于随机游走采样需要访问可能分布在多台机器上的其他节点。与 D 的计算类似，每次会处理 b 个节点，总通信轮次为 $O(\frac{nl^2}{b})$。由于需要使用 $O(bl^2N)$ 的空间来存储与随机游走相关的计算向量，因此总空间复杂度为 $O\left((bl^2\log n)/\epsilon_p^2 + n + m\right)$。

总之，CloudWalker 需要 $O(nl(I+l)/b)$ 轮通信，且总空间复杂度为 $O\left((bl^2\log n)/\epsilon_p^2 + n + m\right)$，其中每台机器需要 $O\left((bl^2\log n)/\epsilon_p^2 + (m+n)/M\right)$ 空间。此外，CloudWalker 需要在每台机器的空间内存开销与通信轮数之间进行权衡。一个特殊情况是，当 b 等于 n 时，CloudWalker 可以在 $O(Il + l^2)$ 轮内完成 SimRank 计算，且每台机器需要 $O(n\log n)$ 空间。最后，CloudWalker 仅在估算 D 和截断长度 l时给出了误差界；他们没有展示计算 SimRank 分数 [14]的任何精度保证。

A.2 UniWalk [28]

UniWalk 的应用范围是无向图。为了计算 SimRank分数 $s(u, v)$，UniWalk 采用蒙特卡罗方法，通过采样一定数量的长度为 L 的随机游走，并计算从两个节点 u 和 v 出发的双向长度为 L 的随机游走的期望相遇距离。UniWalk 的主要贡献是将采样 $O(nN)$ 条双向长度为 L 的游走，转换为仅采样 $O(N)$ 条长度为 $2L$ 的单向游走。引入校正因子，使这两种游走等价。L 在 UniWalk 中根据经验设定，UniWalk 需要 $l = 2L$ 轮次来生成长度为 $2L$ 的单向游走。通过采样 $N = \frac{c^2(1-c^L)^2 d_{\max}^2}{2\epsilon^2(1-c)^2}(2n)\log(\frac{2n}{\delta})^2 d_{\min}^2$ 条随机游走，其中 d_{max} 和 d_{min} 分别是 G 中的最大度和最小度。UniWalk 可以保证以高概率计算出的 SimRank 值的误差至多为 ϵ。在 UniWalk 的分布式版本（[28] 中的算法 3）中，为了保证 $O(l)$ 个通信轮次，我们需要并行执行算法 3 N 次。此时，在最坏情况下，第 21 行会生成 N 次迭代，而第 23 行需要发送大小为 $O(l)$ 的消息。注意，$O(Nl)$ 已经是 UniWalk 过程中产生的最坏情况下的总消息空间开销。因此，每台机器的空间开销和总空间开销均为 $O(Nl) = O(n^2 l\log_{\epsilon^2} n)$，因为 $d_{max} = O(n)$ 且 $d_{min} \geq 1$。

A.3 DISK [14]

与 CloudWalker 类似，DISK 也利用随机游走来估算对角修正矩阵 D，然后执行单源 SimRank 计算。我们也从**计算 D** 和**单源 SimRank 计算**的角度展示这些过程。

计算 D。 </style}以下分析基于 [14] 第 4 节中的结果。DISK 探讨了 D 的物理含义，并提出一种更高效的算法来估算对角修正矩阵 D。具体而言，该方法将 D 的计算转化为从每个节点 $v \in V$ 出发的两条 $\sqrt{c}$ 随机游走的相遇概率。该方法的主要优势是能够保证 D 的估算精度。随后，DISK 构建了一种基于树的方法，在为每个节点 $v \in V$ 设定失败概率 $\delta_d = \frac{1}{n}$ 后，对 $N = O(\log \frac{n}{\epsilon_d^2})$ 棵树进行采样，生成 $(\log \frac{n}{\epsilon_d^2})$ 棵树。这些树也可以解释为 $\sqrt{c}$ 随机游走。由于树的高度期望为 $O(\log n)$，且每棵树的每一层都需要一个通信轮次，因此构建 N 棵树后，总通信轮次为$O(\log^2_{\epsilon_d^2} n)$。在空间开销方面，DISK 需要保存整棵树（随机游走），其空间开销为 $O(\log n)$，每台机器的空间开销则为 $O(\frac{m}{\log_M n})$。

单源 SimRank 的计算。最终的单源计算基于公式 3 的截断形式：

$$\mathbf{S} \approx \sum_{t=0}^{K} c^t \boldsymbol{P}^{\top t} \boldsymbol{D} \boldsymbol{P}^t, \tag{4}$$

其中 K 表示用于近似的截断项。为了获得最终的SimRank分数，将消耗 $O(K)$通信轮次来计算公式4的结果。由于每个节点还需要为上述公式维护 $O(K)$空间，因此每台机器的空间将为 $O(\frac{Kn}{M})$。

因此，我们总结得出，DISK 的总体通信轮次为 $O\left(\log^2_{\epsilon_d^2} n + K\right)$。随后，在空间分析中，DISK 需要总空间 $O(m\log n + Kn)$，并且每台机器需要空间 $O(\frac{m}{\log_M n+Kn})$ 来存储树。DISK的估计误差被控制在小于 $\frac{c(1-c^K)}{1-c}\epsilon_d + c^{K+1}$，且以高概率成立，这与我们在表1中的统计量相对应。

B 缺失的证明

B.1 定理 3 的证明

证明。对于任意实例 n，令 β 满足 $n^\beta = n^\alpha/\log_{\sqrt{c}} n$。如果 $n^\beta > (\log^3 n \cdot \log_{\sqrt{c}} n)$（等价地，$n^\alpha > \log^4 n \log_{\sqrt{c}} \log^5 n \log_{\sqrt{c}})$，那么 $\log_{\sqrt{c}} n = o(S)/\log^3 n$，因此定理 2 适用于 $S = n^\beta$。将定理 2 应用于 $S = n^\beta$，仍然得到 $O(\log^2 \log n + \log^2 l)$ 的轮复杂度和 $O(m + n^{1+o(1)} l^{3.5} + Nl^{2+o(1)})$ 的空间复杂度，因为这两种复杂度都隐藏了 α（或 β）因子。此外，将定理 2 应用于 $S = n^\beta$ 时，机器数量 M 满足 $M \cdot n^\beta = \tilde{\Theta}(m+n)$，这是 MPC 模型的要求。因此，$n^\alpha \cdot M = \tilde{\Theta}(m+n)$ 也成立，这意味着当 $= n^\alpha$ 时，所用机器数量符合 MPC 的要求。现在，对于 M 台机器（每台机器的空间为 n^α），我们将机器空间划分为 $\log_{\sqrt{c}} n$ 个子空间，并对每个子空间分别并行应用 WRME 算法（定理 2），对应于 $l = 1$、2、.....、$\log_{\sqrt{c}} n$。

这表明，每个 WRME 算法的运行长度为 $S = n^\beta$。最坏情况轮数复杂度对应于运行长度最长的 WREM 算法，其长度为 $\log_{\sqrt{c}} n$。根据定理 2，

我们得到轮复杂度

$$O\left(\left\lceil \log^2 \log n + \log^2 l \right\rceil\right) = O\left(\left\lceil \log^2 \log n + \log^2 \log_{\frac{1}{\sqrt{c}}} n \right\rceil\right) = O\left(\log^2 \log n\right) \tag{5}$$

最后一步成立，因为 c 是常数。对于每个 WRME 实例，机器空间 n^β 就足够了。考虑所有并行运行的 $\log_{\frac{1}{\sqrt{c}}} n$ 个 WRME 实例，每台机器所需空间为 $n^\beta \cdot \log_{\frac{1}{\sqrt{c}}} n = n^\alpha$。

对于总空间开销，我们需要将每次调用 WRME 算法的空间开销全部相加。因此，通过对 $1 \le l \le \log_{\frac{1}{\sqrt{c}}} n$，以及对于每个 l，空间开销会扩展至 $\frac{2n}{(\epsilon - \frac{3}{n})^2} \cdot (\sqrt{c})^l \cdot (1-\sqrt{c})$ 倍（随机游走的数量）：

$$\begin{aligned} &\sum_{l=1}^{\log_{\frac{1}{\sqrt{c}}} n} O\left(m + n^{1+o(1)} l^{3.5} + N_l n l^{2+o(1)}\right) \\ =& \sum_{l=1}^{\log_{\frac{1}{\sqrt{c}}} n} O\left(m + n^{1+o(1)} l^{3.5} + \frac{\log 2n}{2(\epsilon - \frac{3}{n})^2} \cdot (\sqrt{c})^l \cdot (1-\sqrt{c}) n l^{2+o(1)}\right) \\ =& O\left(m \log n + n^{1+o(1)} \log^{4.5} n + \frac{n \log^{3+o(1)} n}{2(\epsilon - \frac{3}{n})^2}\right) \end{aligned} \tag{6}$$

最后一步成立，因为 $\sum_{l=1}^{\log_{\frac{1}{\sqrt{c}}} n} (\sqrt{c})^l (1-\sqrt{c}) < 1$。每台机器的空间开销相较于定理2所需的每台机器开销至多增加 $\log_{\frac{1}{\sqrt{c}}} n$，因此仍然远低于 n。□

B.2 引理 1 的证明

证明。考虑存在一个分布式算法 A，它在 $M = \tilde{O}(m+n)$ 台机器上运行，每台机器的空间为 $S = \Theta(n^\alpha)$，用于 $0 < \alpha < 1$，并使用 $O(g(\alpha) \cdot f(n))$ 个通信轮次，其中 $g(\alpha)$ 是基于 α 计算得出的常数函数。根据大-O 记号的定义，必定存在常数 C、C' 和 n_0，使得对于任意 $n' > n_0$，A 使用的每台机器的空间 S 至多为 $C' \cdot n'^\alpha$，且 A 使用的轮次数至多为 $C' \cdot g(\alpha) \cdot f(n')$。

令 $\beta = \log_{n'}(C' \cdot n'^\alpha)$，则有 $n'^\beta = C' \cdot n'^\alpha$。因此，算法 A 在空间为 n'^β 的机器上执行，所用轮次至多为 $C' \cdot g(\alpha) \cdot f(n') = \tilde{O}(1) \cdot f(n')$。机器数量为 $M = \tilde{O}(\frac{m+n}{n'^\alpha}) \cdot (C' n'^\alpha) \cdot (\frac{m+n}{n'^\beta})$。因此，当图节点数量为 n' 时，算法 A 是适用于 $S = n'^\beta$ 的有效大规模并行计算算法，且所用轮次为常数因子乘以 $f(n')$。

固定 β，并令 M^* 为考虑所有 $n' > n_0$ 时所用机器数量的最大值，不难看出 $M = \tilde{O}(\frac{m+n}{n^\beta})$。

同时，注意到 $\beta = \alpha + \log_{n'}(C')$。当 n' 足够大时，β 可以任意接近 α。因此，对于任意 $\beta \in (0,1)$，当 n' 足够大时，我们总能找到 $\alpha = \beta - \log_{n'}(C') > 0$，从而构造出相应的大规模并行计算算法 A。

然后我们就可以放心地将符号 β 替换为 α，引理便成立。□

B.3 引理 3 的证明

证明。假设游走W和游走W'在节点 v处的第 i步相遇，那么存在一个来自W的分解元组 $T = (j, i, v, l_1, s)$，以及一个来自W'的分解元组 $T' = (j, i, v, l_2, u)$。它们共享前三个元素。此外，我们通过收缩证明，存在一个源自W的分解元组 T^*，但 T^* 在排序顺序中更接近 T'。由于 T和 T'的前三个元素相同，因此 T^* 也必须具有相同的前三个元素。然而， T和 T^* 都由游走W分解而来，但从同一个游走分解出的任意两个元组都必须在元组的前三个位置具有不同的元素。矛盾由此产生。□

B.4 定理 4 的证明

证明。该证明直接由定理 3 以及第 5 节中的所有分析结果得出。□

B.5 定理 5 的证明

我们的目标是展示，我们的算法能够获得与真实值相比误差至多为 ϵ 的 SimRank 值。回顾一下，对于每个节点，我们采样 $N_l = \left\lceil \frac{2n}{2\epsilon - \frac{3}{n}^2} \sqrt{c}^{\,l} \left(1 - \sqrt{c}\right) \right\rceil$ 个长度为 $l \le \log_{\frac{1}{\sqrt{c}}}(\cdot)$ 的游走。我们将通过两个小节来说明如何界定估计误差。

B.5.1 控制截断误差。基于蒙特卡洛方法，我们将展示，通过从 s和 u采样截断的 $\sqrt{c}$-衰减游走来检查相遇概率，可以得到一个 $\epsilon (\epsilon \ge \frac{3}{n})$ 的绝对误差。推导基于如下霍夫丁不等式：

引理 4. 设 $X_1, \ldots, X_z$ 是相互独立的有界随机变量，且对所有 i，均有 $X_i \in [0,1]$。那么

$$\mathbf{Pr}\left[\left|\frac{1}{z}\sum_{i=1}^{z}(X_i - E[X_i])\right| \ge \lambda\right] \le 2e^{(-2z\lambda^2)}. \tag{7}$$

注意，从节点 s 和节点 u 出发的一对 $\sqrt{c}$ 游走相遇的概率就是 SimRank 值 $s(s, u)$，并令 X_i 表示从 s 和 u 出发的第 i 对 $\sqrt{c}$ 游走相遇这一事件。因此，$\mathbb{E}[X_i] = s(s, u)$。由霍夫丁不等式以及估计 $\tilde{s}(s, u) = \frac{\sum_{i=1}^{z} X_i}{z}$，简单可得

$$\begin{aligned} &\mathbf{Pr}\left[|\tilde{s}(s,u) - s(s,u)| \ge \lambda\right] \\ =&\mathbf{Pr}\left[\left|\frac{\sum_{i=1}^{z} X_i}{z} - \frac{\sum_{i=1}^{z} E(X_i)}{z}\right| \ge \lambda\right] \le 2e^{-2z\lambda^2} \end{aligned} \tag{8}$$

令 $z = \log_{2\lambda^2} \frac{2n}{n}$，我们有 $e^{-2z\lambda^2} = \frac{1}{n}$。然而，我们的算法不涉及长度超过 $\log_{\frac{1}{\sqrt{c}}} n$ 的 $\sqrt{c}$-游走。为量化截断的影响，我们注意到，采样a $\sqrt{c}$-衰减游走等价于以概率 $(\sqrt{c})^l(1-\sqrt{c})$ 采样长度为 l 的游走，其中 $l \ge 0$。通过计算其中 a

当 $\sqrt{c}$-游走小于 $\log_{\frac{1}{\sqrt{c}}} n$ 时，我们有：

$$\mathbf{Pr}\left[l \le \log_{\frac{1}{\sqrt{c}}} n\right] = \sum_{l=0}^{\log_{\frac{1}{\sqrt{c}}} n} (\sqrt{c})^l \cdot (1-\sqrt{c}) > 1 - \frac{1}{n} \tag{9}$$

上述方程表明，长度超过 $\log_{\frac{1}{\sqrt{c}}} n$ 的随机游走的概率小于 $\frac{1}{n}$。因此，我们根据这个概率结果设置长度为 l（$0 \le l \le \log_{\frac{1}{\sqrt{c}}} n$）的 $N_l = \left\lceil \log_{2\lambda^2} 2n \cdot (\sqrt{c})^l \cdot (1-\sqrt{c}) \right\rceil$，并将长度截断为 $\log_{\frac{1}{\sqrt{c}}} n$。接下来我们将证明，这种简化对 SimRank 值的绝对值影响小于 $\frac{3}{n}$。

我们将 C_{s,l_1,u,l_2} 表示 s 的长度为 l_1 的游走与 u 的长度为 l_2 的游走之间的相遇次数，若我们基于期望数量直接执行随机游走，即 $|Z_{s,l}|$。令 $\phi = \log_{1/\sqrt{c}} n$。我们可以将公式 8 转换如下：

$$\begin{aligned}
&\mathbf{Pr}\left[|\tilde{s}(s,u) - s(s,u)| < \lambda\right] \\
=&\mathbf{Pr}\left[-\lambda < \frac{\sum_{l_1=0}^{+\infty}\sum_{l_2=0}^{+\infty} C_{s,l_1,u,l_2}}{z} - s(s,u) < \lambda\right] \\
=&\mathbf{Pr}\left[-z\lambda < \sum_{l_1=0}^{\phi}\sum_{l_2=0}^{\phi} C_{s,l_1,u,l_2} + \sum_{l_1=0}^{\phi}\sum_{l_2=\phi+1}^{+\infty} C_{s,l_1,u,l_2} + \right. \\
&\left. \sum_{l_1=\phi+1}^{+\infty}\sum_{l_2=0}^{\phi} C_{s,l_1,u,l_2} + \sum_{l_1=\phi+1}^{+\infty}\sum_{l_2=\phi+1}^{+\infty} C_{s,l_1,u,l_2} - s(s,u)\cdot z < z\lambda\right] \\
\ge& 1 - 2e^{-2z\lambda^2}
\end{aligned} \tag{10}$$

根据公式 9，起点为节点 s（分别为 u）且长度超过 $\phi = \log_{1/\sqrt{c}} n$ 的随机游走少于 $\frac{z}{n}$ 条。因此，当一条游走的长度超过 $\phi = \log_{1/\sqrt{c}} n$ 时，配对游走的相遇次数至多为 $\frac{z}{n}$。形式化地，

$$\sum_{l_1=0}^{\phi}\sum_{l_2=\phi+1}^{+\infty} C_{s,l_1,u,l_2} \le \frac{z}{n}$$

$$\sum_{l_1=\phi+1}^{+\infty}\sum_{l_2=0}^{\phi} C_{s,l_1,u,l_2} \le \frac{z}{n}$$

$$\sum_{l_1=\phi+1}^{+\infty}\sum_{l_2=\phi+1}^{+\infty} C_{s,l_1,u,l_2} \le \frac{z}{n}$$

于是我们有

$$\begin{aligned}
0 <& \frac{\sum_{l_1=1}^{\phi}\sum_{l_2=\phi+1}^{+\infty} C_{s,l_1,u,l_2}}{z} + \frac{\sum_{l_1=\phi+1}^{+\infty}\sum_{l_2=1}^{\phi} C_{s,l_1,u,l_2}}{z} \\
&+ \frac{\sum_{l_1=\phi+1}^{+\infty}\sum_{l_2=\phi+1}^{+\infty} C_{s,l_1,u,l_2}}{z} < \frac{3}{n}.
\end{aligned} \tag{11}$$

如果我们将公式 11 应用于公式 10，则有：

$$\begin{aligned}
&\mathbf{Pr}\left[|\tilde{s}(s,u) - s(s,u)| < \lambda\right] \\
\le&\mathbf{Pr}\left[-\lambda - \frac{3}{n} < \frac{\sum_{l_1=0}^{\phi}\sum_{l_2=0}^{\phi} C_{s,l_1,u,l_2}}{z} - s(s,u) < \lambda\right] \\
\le&\mathbf{Pr}\left[-\lambda - \frac{3}{n} < \frac{\sum_{l_1=0}^{\phi}\sum_{l_2=0}^{\phi} C_{s,l_1,u,l_2}}{z} - s(s,u) < \lambda + \frac{3}{n}\right]
\end{aligned} \tag{12}$$

由于我们只测量游走长度小于 $\phi = \log_{1/\sqrt{c}} n$ 时发生的相遇，因此我们将 $\tilde{s}(s,u) = \frac{\sum_{l_1=0}^{\phi}\sum_{l_2=0}^{\phi} C_{s,l_1,u,l_2}}{z}$ 设为我们的估计结果。最后，我们得到如下精度保证：

$$\mathbf{Pr}\left[|\tilde{s}(s,u) - s(s,u)| < \lambda + \frac{3}{n}\right] \ge 1 - 2e^{-2z\lambda^2} \tag{13}$$

令 $\lambda = \epsilon - \frac{3}{n}$，即可将该结果简化为：

$$\mathbf{Pr}\left[|\tilde{s}(s,u) - s(s,u)| \ge \epsilon\right] \le 2e^{-2z(\epsilon - \frac{3}{n})^2}, \tag{14}$$

其中 $z = \log 2n$ 且 $\epsilon \ge \frac{3}{n}$。基于此结果，我们保证：对于每个节点，采样 $N_l = \left\lceil \log 2n \cdot (\epsilon - \frac{3}{n})^2 \cdot (\sqrt{c})^l \cdot (1-\sqrt{c}) \right\rceil$ 长度为-l($0 \le l \le \log_{\frac{1}{\sqrt{c}}}$) 的游走，即可获得误差至多为 ϵ（$\epsilon \ge \frac{3}{n}$）的 SimRank 值 $\tilde{s}(s,u)$。

B.5.2 向上取整误差。我们认为，对于每个长度 l，随机游走的实际数量为 $\left\lceil \log 2n \cdot (\epsilon - \frac{3}{n})^2 \cdot (\sqrt{c})^l \cdot (1-\sqrt{c}) \right\rceil$，因为 $\log 2n \cdot (\epsilon - \frac{3}{n})^2 \cdot (\sqrt{c})^l \cdot (1-\sqrt{c})$ 可能不是整数。对长度为 l 的样本进行向上取整会使我们的 SimRank 估计产生偏差，下面通过一个例子对此进行说明。令 $\hat{Z}_{s,l_1}$ 表示从 s 出发实际生成的长度为 l_1 的随机游走集合，其中 $|\hat{Z}_{s,l_1}| = \left\lceil \log 2n \cdot (\epsilon - \frac{3}{n})^2 \cdot (\sqrt{c})^{l_1} \cdot (1-\sqrt{c}) \right\rceil$。此外，令 Z_{s,l_1} 表示长度为 l_1 的随机游走的期望集合，其中 $|Z_{s,l_1}| = \log 2n \cdot (\epsilon - \frac{3}{n})^2 \cdot (\sqrt{c})^{l_1} \cdot (1-\sqrt{c})$。当从 Z_{s,l_1} 和 Z_{u,l_2} 中选出的两条随机游走相遇时，将 $|Z_{s,l_1}|$ 向上取整为 $|Z_{s,l_1}|$，并将 $|Z_{u,l_2}|$ 向上取整为 $|\hat{Z}_{u,l_2}|$，会改变游走长度分布。因此，s 和 u 的相遇概率会受到影响，从而降低我们的精度。结果如下：

引理 5。每当从 $\hat{Z}_{s,\,l_1}$ 和 $\hat{Z}_{u,\,l_2}$ 中选取的两次随机游走相遇时，此次相遇仅会为 $\frac{|Z_{s\,l_1}|}{|Z_{s\,l_1}|} \cdot \frac{|Z_{u\,l_2}|}{|Z_{u\,l_2}|} \cdot \frac{1}{z}$ 的 *SimRank* 值 $\tilde{s}$（s，u）贡献分数。

证明。当我们基于变更后的样本数量 $|Z_{s,\,l_1}|$ 和 $|Z_{u,\,l_2}|$ 估计 SimRank $\tilde{s}(s,\,u)$ 值时，我们仍需确保相遇概率与使用期望样本数量 $|Z_{s,\,l_1}|$ 和 $|Z_{u,\,l_2}|$ 时相同：

$$\frac{\hat{C}_{s,l_1,u,l_2}}{|\hat{Z}_{s,l_1}| \cdot |\hat{Z}_{u,l_2}|} = \frac{C_{s,l_1,u,l_2}}{|Z_{s,l_1}| \cdot |Z_{u,l_2}|}. \tag{15}$$

ˆHereC_{s,l_1,u,l_2} 表示实际运行随机游走时， s的长度为l_1的游走与 u的长度为l_2 的游走相遇的实际相遇次数。ˆC_{s,l_1,u,l_2} 与C_{s,l_1,u,l_2} 之间的区别在于，C_{s,l_1,u,l_2}表示运行理论随机游走时的相遇次数。然后，我们根据以下公式推导 SimRank 值 $\tilde{s}(s,u)$:

$$\tilde{s}(s,u) = \frac{\sum_{l_1=1}^{\log_{\frac{1}{\sqrt{c}}} n} \sum_{l_2=1}^{\log_{\frac{1}{\sqrt{c}}} n} C_{s,l_1,u,l_2}}{z} = \frac{\sum_{l_1=1}^{\log_{\frac{1}{\sqrt{c}}} n} \sum_{l_2=1}^{\log_{\frac{1}{\sqrt{c}}} n} \hat{C}_{s,l_1,u,l_2} \cdot \frac{|Z_{s,l_1}| \cdot |Z_{u,l_2}|}{|\hat{Z}_{s,l_1}| \cdot |\hat{Z}_{u,l_2}|}}{z} \tag{16}$$

ˆ这里C_{s,l_1,u,l_2} 是通过统计来自实际样本集的游走 ˆZ_{s,l_1} 和 ˆZ_{u,l_2} 相遇的次数得出的。该结果证明了这一论断。

□

B.6 降低阈值 $\frac{3}{n}$ 并降低失败概率 $\frac{1}{n}$

在定理 5 中，我们注意到，通过分别增加游走长度和样本数量，可以进一步降低误差阈值（例如， $\frac{3}{n}$）和失败概率（例如， $\frac{1}{n}$）。

降低误差阈值。 假设我们将最大长度扩展至 $p \cdot \log_{\sqrt{c}} n$，其中 p 是一个常数，且 $p \geq 1$，那么我们可以将公式 9 变换如下:

$$\Pr\left[l \leq p \cdot \log_{\frac{1}{\sqrt{c}}} n\right] = \sum_{l=0}^{p \cdot \log_{\frac{1}{\sqrt{c}}} n} (\sqrt{c})^l \cdot (1-\sqrt{c}) \tag{17}$$

$$= 1 - \sqrt{c}^{\,p \cdot \log_{\frac{1}{\sqrt{c}}} n + 1} > 1 - \frac{1}{n^p} \tag{18}$$

因此，长度超过 $p \log_{\sqrt{c}} n$ 的随机游走的概率小于 $\frac{1}{n^p}$。此外，公式 11 中的上界可以被$\frac{3}{n^p}$，而我们在定理 5 中的误差变为 ϵ ($\epsilon \geq \frac{3}{n^p}, p \geq 1$)。

降低失败概率。失败概率由公式 14 推导而来。我们可以通过采样更多随机游走来简单地降低失败概率。特别地，在误差阈值$\frac{3}{n^p}$ 下，通过设置 $z = (\log 2n^q)/2\left(\epsilon - \frac{3}{n^p}\right)^2$，其中 q 是常数且 $q \geq 1$。我们可以将公式 14 转换如下:

$$\Pr\left[|\tilde{s}(s,u) - s(s,u)| \geq \epsilon\right] \leq 2e^{-2z(\epsilon - \frac{3}{n^p})^2} = \frac{1}{n^q} \tag{19}$$

总之，通过将最大长度设置为 $p \log_{\sqrt{c}} n$ 并采样 $z = (\log 2n^q)/2\left(\epsilon - \frac{3}{n^p}\right)^2$，我们可以保证 $\epsilon(\epsilon \geq \frac{3}{n^p})$误差，并以 $1 - \frac{1}{n^q}$ 概率成立，其中 $p, q \geq 1$。显然，这种转换只会将随机游走的数量和游走长度增加一个常数因子，不会影响轮复杂度和空间复杂度。例如，将样本数量增加到$(\log 2n^2)/2\left(\epsilon - \frac{3}{n^2}\right)^2$并将最大长度延长至$2 \log_{\sqrt{c}} n$，我们可以保证 $\epsilon(\epsilon \geq \frac{3}{n^2})$误差，并以 $1 - \frac{1}{n^2}$ 概率成立。