计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202440148
61(8): 1865−1881, 2024

动态有权图上的随机游走概率计算

王涵之[1] 易 璐[2] 魏哲巍[2,3,6,7] 甘骏豪[4] 袁 野[5] 文继荣[1,2,6] 杜小勇[1,7]

[1](中国人民大学信息学院 北京 100872)

[2](中国人民大学高瓴人工智能学院 北京 100872)

[3](琶洲实验室(黄埔) 广州 510555)

[4](墨尔本大学 澳大利亚墨尔本 VIC3010)

[5](北京理工大学计算机学院 北京 100081)

[6](大数据管理与分析方法研究北京市重点实验室(中国人民大学高瓴人工智能学院) 北京 100872)

[7](数据工程与知识工程教育部重点实验室(中国人民大学) 北京 100872)

(hanzhi_wang@ruc.edu.cn)

Random-Walk Probability Computation on Dynamic Weighted Graphs

Wang Hanzhi[1], Yi Lu[2], Wei Zhewei[2,3,6,7], Gan Junhao[4], Yuan Ye[5], Wen Jirong[1,2,6], and Du Xiaoyong[1,7]

[1](*School of Information*, *Renmin University of China*, *Beijing* 100872)

[2](*Gaoling School of Artificial Intelligence*, *Renmin University of China*, *Beijing* 100872)

[3](*Pazhou Laboratory* (*Huangpu*), *Guangzhou* 510555)

[4](*University of Melbourne, Melbourne, Australia,* VIC3010)

[5](*School of Computer Science and Technology*, *Beijing Institute of Technology*, *Beijing* 100081)

[6](*Beijing Key Laboratory of Big Data Management and Analysis Methods* (*Gaoling School of Artificial Intelligence*, *Renmin University of China*), *Beijing* 100872)

[7](*Key Laboratory of Data Engineering and Knowledge Engineering* (*Renmin University of China*), *Ministry of Education*, *Beijing* 100872)

Abstract Computing random-walk probabilities on graphs is the subject of extensive research in both graph theory and data mining research. However, existing work mainly focuses on static graphs, and cannot efficiently support dynamic weighted graphs, which are ubiquitous in real-world applications. We study the problem of computing random-walk probabilities on dynamic weighted graphs. We propose to use a sampling schema called coin flip sampling, rather than the more commonly adopted weighted sampling schema, for simulating random walks in dynamic weighted graphs. We demonstrate that simulations based on coin-flip sampling maintain the unbiasedness of the resulting random-walk probability approximations. Moreover, this approach allows us to simultaneously achieve a near-optimal query time complexity and an optimal $O(1)$ update time overhead per edge insertion or deletion. This is a significant improvement over existing methods, which typically incur substantial sampling costs or rely on intricate

收稿日期: 2024-03-02; **修回日期**: 2024-04-25

基金项目: 国家自然科学基金项目(U2241212, 61932001); 北京市自然科学基金项目(4222028); 北京高校卓越青年科学家项目(BJJWZYJH012019100020098); 华为下一代智能信息分发技术研究项目; 中国人民大学2023年度拔尖创新人才培育资助计划项目; 中央高校建设世界一流大学(学科)和特色发展引导专项资金; 新一代智能搜索与推荐教育部工程研究中心资助; 中国人民大学大型科学仪器共享平台资助

This work was supported by the National Natural Science Foundation of China (U2241212, 61932001), the Beijing Natural Science Foundation (4222028), the Beijing Outstanding Young Scientist Program (BJJWZYJH012019100020098), the Huawei-Renmin University Joint Program on Information Retrieval, the Outstanding Innovative Talents Cultivation Funded Programs 2023 of Renmin Univertity of China, the Fund for Building World-class Universities (disciplines) of Renmin University of China, the Engineering Research Center of Next-Generation Intelligent Search and Recommendation Funds, Ministry of Education, and the Public Computing Cloud, Renmin University of China.

通信作者: 魏哲巍(zhewei@ruc.edu.cn)

auxiliary structures that are hard to maintain in a dynamic setting. We present both theoretical analysis and empirical evaluations to substantiate the superiority of our method on dynamic weighted graphs.

Key words random-walk probability computation; dynamic weighted graphs; coin-flip sampling; real-time update; large-scale graphs

摘 要 图上的随机游走概率计算是传统图论与现代数据挖掘领域普遍关注的问题之一.现有工作普遍关注静态图上的随机游走概率计算,却鲜少关注与实际应用场景更贴合的权重动态图.针对动态有权图上的随机游走概率计算问题,提出了一种基于硬币翻转采样的随机游走概率计算方法.相比于传统的基于权重采样的随机游走概率计算方法,所提方法可以在保证随机游走概率计算结果无偏的前提下,同时做到近似最优的随机游走概率计算复杂度和最优的采样结构更新复杂度.作为对比,现有方法或具有较大的计算时间复杂度,或依赖于复杂的索引结构而难以在动态图上即时更新.对所提方法做出了详细的理论分析,并在真实图数据集上进行模拟实验,实验结果证实了所提方法的有效性.

关键词 随机游走概率计算;动态有权图;硬币翻转采样;实时更新;大规模图

中图法分类号 TP391

图(graph)是计算机科学中一类重要的数据结构,由节点和连接这些节点的边构成.在图上进行随机游走(random walks on graphs)是一种常用的图信息挖掘方式,所述随机游走本质上是一条马尔可夫链(Markov chain),随机游走在各步走到的节点对应马尔可夫链在各时刻的状态.

具体来说,给定随机游走的源节点s,假设该游走在当前时刻走到节点u(在起始时刻u即为s),在下一时刻,该游走会从节点u的出邻居集合中随机选取一个节点v,从节点u走到节点v.随机游走选取节点v的概率与边(u,v)的权重有关.在无权图上,图上所有边的权重为1.因此,随机游走会从节点u的出邻居集合中均匀随机地选取一个节点作为节点v.若在有权图上,图上各边带有边权,随机游走则会按照节点u出边的边权分布选择节点v.具体而言,如果用d_u表示节点u的出度,用A_{uv}表示边(u,v)的权重,则若一条随机游走在当前时刻走到节点u,其在下一时刻会以$\frac{A_{uv}}{d_u}$的概率从节点u的所有出邻居中选出节点v,从节点u走到节点v.

计算图上的一条随机游走在其游走各步走到各节点的概率被称为随机游走概率的计算(random-walk probability computation)问题,在可控误差下进行随机游走概率的近似计算是图论、数据挖掘、网络分析等领域的核心研究问题,相关技术在图节点邻近度计算[1-7]、图神经网络[8-10]、社区发现[11-12]等领域均有广泛应用.

本文关注动态有权图上随机游走概率的近似计算问题.此处,动态有权图指图上各边带有权重、图结构会随时间动态变化的图数据类型.动态有权图在实际应用场景中极为常见.例如,在社交网络中,社交用户与图节点对应,用户之间的聊天关系与图节点间的连边相对应,用户之间联系的频繁程度可被量化为边权,用户的注册与注销行为可由图结构上节点数的增减反映.因此,真实应用场景中的社交网络可由动态有权图来刻画.类似地,道路交通网络也可用动态有权图刻画,其中站点对应图节点,站点之间的通达关系对应节点间的连边,站点间的距离反映为边权,站点和道路的增减反映为图结构的动态变化.

尽管动态有权图可以更好地建模真实网络结构,但由于其上随机游走概率计算的复杂性,现有方法大多面向静态无权图,而在动态有权图上的执行效率不高.具体而言,现有计算随机游走概率的方法大多是蒙特卡洛方法(Monte-Carlo method),即通过在图上产生大量的随机游走来完成图上随机游走概率的近似估计.因此,随机游走概率的计算效率和各条随机游走的生成效率直接相关,而生成一条随机游走的时间取决于在游走各步从当前节点的出邻居集合中采样节点的时间.在静态无权图上,借助简单的均匀采样(uniform sampling)技术即可在$O(1)$时间内完成1步随机游走采样.但在动态有权图上,随机游走采样需借助特定的采样技术方能高效完成.现有方法普遍调用权重采样(weighted sampling)技术进行随机游走采样,但受限于权重采样的采样模式设定,其在动态有权图上的采样效率不高,且普遍依赖于复杂的采样结构.这使得现有计算随机游走概率的方法在动态有权图上或计算效率低下,或受制于复

杂的采样结构而难以实时处理动态图的结构变化. 如何在动态有权图上高效完成随机游走概率的近似计算，虽经多年研究但仍未被解决.

针对上述问题，本文提出将硬币翻转采样（coin flip sampling）技术应用至动态有权图上的随机游走概率问题中. 本文证明，相较于既往借助权重采样技术在动态有权图上生成随机游走的方法，基于硬币翻转采样的方法更具灵活性，无需复杂的采样结构即可在动态有权图上同时实现随机游走概率的高效计算和实时更新.

具体而言，本文首先为现有硬币翻转采样方法设计了一种支持动态图上实时更新的采样结构，该采样结构可以在不影响现有硬币翻转采样方法采样效率的前提下，实时处理动态有权图上的边增删操作.

进一步地，本文将硬币翻转采样方法应用至动态有权图上的随机游走概率计算中，提出了一个新颖的算法 CoinFlipWalk. 本文证明，CoinFlipWalk 算法可以在保证随机游走概率计算结果无偏性的前提下，同时达到近似最优的随机游走概率计算复杂度和最优的采样结构更新复杂度. 特别地，CoinFlipWalk 算法同时支持无权图上的随机游走概率计算，可以在最优的时间复杂度下完成计算.

最后，本文在多个大规模动态有权图上进行了模拟实验. 实验结果显示，在达到同级别近似计算误差的前提下，CoinFlipWalk 算法的运行时间和更新时间均小于既往方法，这证实了 CoinFlipWalk 算法相较于既往方法的优越性.

1 预备知识

1.1 符号和计算模型

本文使用$G=(V,E)$表示所关注的有权动态图结构，其中V指代图G的节点集，E指代边集. 本文用$n=|V|$表示图G的节点数，用$m=|E|$表示图G的边数. 对于图上任意节点u，本文用$N_{\text{in}}(u)$和$N_{\text{out}}(u)$分别表示节点u的入邻居节点集合和出邻居节点集合；用$n_u=|N_{\text{out}}(u)|$表示节点u出邻居的个数. 此外，本文用$\boldsymbol{A}$表示图G的邻接矩阵，用A_{uv}表示边(u,v)的权重，用$d_u=\sum_{v\in N_{\text{out}}(u)}A_{uv}$表示节点$u$的出度，用$A_u^*$表示节点$u$出边边权的最大值.

本文使用 word RAM 模型分析算法的时间复杂度. Word RAM 模型是一个常用的计算模型，用以模拟真实的计算机运行情况. Word RAM 模型的基本假设是：一个可随机存取的机器（random access machine）可以对一个由w位组成的字进行位操作，且机器寻址大小（即2^w）与问题规模大小相匹配. 需要注意的是，随机游走概率计算问题的问题规模大小为$O(n+m)=O(n^2)$. 因此，为遵从 Word RAM 模型的基本假设，本文亦假设所用数值（例如，任一边权A_{uv}）都可以用$O(\log n)$比特来表示. 此外，为了表述与分析方便，本文假设图G上各边权均满足$A_{uv}\geqslant 1$. 对于任意给定图G，在读图时对其上各边的边权A_{uv}进行统一处理$A_{uv}\leftarrow A_{uv}/w_{\min}$，在随机游走概率计算全部结束后再进行逆处理$A_{uv}\leftarrow w_{\min}\cdot A_{uv}$，即可保证$A_{uv}\geqslant 1$假设成立且不改变给定图$G$的任何性质. 此处$w_{\min}$指图$G$上各边边权的一个下界.

1.2 随机游走概率的定义与性质

给定图G上的节点s作为随机游走的源节点，给定正整数L作为随机游走的长度，从节点s出发的随机游走在其第L步走到图上各节点的随机游走概率向量$\boldsymbol{\pi}_s^{(L)}$的定义为：

$$\boldsymbol{\pi}_s^{(L)}=\boldsymbol{P}^L\cdot\boldsymbol{e}_s, \tag{1}$$

其中，$\boldsymbol{P}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D}^{-1}$被称为随机游走的转移概率矩阵，$\boldsymbol{e}_s$是一个$n$维的独热向量（one-hot vector），其只有第s维是 1，其余维为 0. 对于图上任意节点u，$\pi_s^{(L)}(u)$即表示从源点s处出发的随机游走在其第L步走到节点u的概率. 为了表述的简洁，本文将在不引起歧义的前提下默认随机游走的源节点为s，并省略$\boldsymbol{\pi}_s^{(L)}$和$\pi_s^{(L)}(u)$的下脚标s，将其简化为$\boldsymbol{\pi}^{(L)}$和$\pi^{(L)}(u)$.

由式（1）中随机游走概率向量可得：$\boldsymbol{\pi}^{(0)}=\boldsymbol{e}_s$，即从源点$s$出发的随机游走在其第 0 步只能走到节点$s$. 对于任意$\ell\in\{1,2,\cdots\}$，从源节点$s$出发的随机游走在其第$\ell$步走到任意图节点$v$的概率$\pi^{(\ell)}(v)$满足关系式：

$$\pi^{(\ell)}(v)=\sum_{u\in N_{\text{in}}(v)}\pi^{(\ell-1)}(u)\cdot\frac{A_{uv}}{d_u}. \tag{2}$$

式（2）表示从节点s出发的随机游走在其第ℓ步走到节点v的概率可被拆分为 2 项：该游走在其第$\ell-1$步走到图节点u的概率，即$\pi^{(\ell-1)}(u)$和该游走在其最后一步由节点u走到节点v的概率，即A_{uv}/d_u. 式（2）可被进一步拓展为关系式：

$$\pi_s^{(\ell)}(v)=\sum_{u\in V}\pi^{(\ell-i)}(u)\cdot\pi_u^{(i)}(v), \tag{3}$$

其中，$\ell\in\{0,1,\cdots\}$，$i\in\{0,1,\cdots,\ell\}$.

1.3 动态有权图上的随机游走概率计算

本文关注动态有权图上随机游走概率的近似计算问题，该问题的具体定义有：

定义 1. 给定动态有权图$G=(V,E)$、源节点$s\in V$、随机游走步长L、相对误差参数c、相对误差阈值δ、失败概率参数p_f，本文期望对图上所有节点u的随机游走概率$\pi^{(L)}(u)$进行近似估计，所得估计值$\hat{\pi}^{(L)}(u)$以不小于$1-p_f$的概率满足：

$$|\pi^{(L)}(u)-\hat{\pi}^{(L)}(u)|\leqslant c\cdot\max\left\{\pi^{(L)}(u),\delta\right\}.$$

为了表述和分析的方便，本文假设相对误差c和失败概率参数p_f均为常数，既往多个工作中亦作此假设.

本文期望设计近似算法以较小的计算复杂度完成定义 1 所要求的近似计算问题. 同时，本文关注所提近似算法所依赖的采样结构在图结构动态变化后的更新时间复杂度.

2 现有工作

图上随机游走由于其刻画节点相关性的能力，在图算法、图机器学习领域有广泛的研究和应用. 近年来，随着研究的深入，有权图和动态图逐渐引起研究者的重视，对有权图和动态图上随机游走应用的工作也有了初步的探究. 例如，郭佳雯等人[13]提出利用随机游走计算动态异质图的网络嵌入，张小驰等人[14]针对子图查询问题提出基于半 Markov 随机游走的算法，Wu 等人[15]提出利用随机游走来辅助计算有权图的嵌入. 除了本文提出的近似算法之外，如何利用矩阵运算精确计算随机游走概率也是一个重要的问题. 例如，Jung 等人[16]提出利用分块求解的方法计算带重启的随机游走(random walk with restart)概率. 总的来说，关于有权图和动态图随机游走的研究还处于初步阶段，仍存在大量有价值的问题等待探索.

本节其余部分将对随机游走概率计算的蒙特卡洛方法作以介绍. 此外，本节对权重采样和硬币翻转采样的采样模式和代表性算法作以概述. 表 1 展示了采样算法在有权图、无权图上单次采样和更新所需的时间开销.

Table 1 Time Complexities for Each Sampling Method at Neighbor of Sampling Node u

表 1 各采样方法在采样节点u的邻居的时间复杂度

采样方法	有权图上的单次采样时间	无权图上的单次采样时间	更新时间开销
前缀和采样[17]	$O(1+\log n_u)$	$O(1+\log n_u)$	$O(1+\log n_u)$
别名采样[18-19]	$O(1)$	$O(1)$	$O(n_u)$
拒绝采样[20]	$O\left(n_u\cdot\frac{A_u^*}{d_u}\right)\star$	$O(1)$	$O(1+\log\log n_u)$
硬币翻转采样	$O(1+\log n_u)\star$[21]	$O(1)\star$[22-23]	$O(1)$（本文）

注：$\star$ 表示期望时间复杂度.

2.1 随机游走概率计算的蒙特卡洛方法

现有计算随机游走概率的方法大致可以分为两类：蒙特卡洛方法(Monte Carlo method)和确定性概率传播方法(deterministic push method). 但是，确定性概率传播方法无法保证计算结果满足估计误差. 故本文仅关注随机游走概率计算的蒙特卡洛方法，对于确定性概率传播方法暂不多作讨论.

蒙特卡洛方法[24]是解决随机问题的经典方法，其基本思想是重复采样大量样本，将各样本中目标结果出现的次数占总采样次数的比例作为目标结果的估计值. 随机游走概率计算的蒙特卡洛方法会重复地在给定图结构上的给定源点s处生成随机游走. 对于图上任意节点u，蒙特卡洛方法将记录所有随机游走中在第L步走到图上节点u的随机游走条数，用该条数占随机游走总条数的比例作为$\pi^{(L)}(u)$的估计值.

算法 1 是用于随机游走概率计算的蒙特卡洛方法的伪代码. 给定动态有权图$G=(V,E)$、源节点s、步长L、随机游走条数n_r，算法 1 会在图G上重复产生n_r条L步随机游走，每条随机游走均从节点s出发，算法 1 记录所有n_r条随机游走中在第L步走到图上各节点(例如节点u)的条数(例如$n_r(u)$)，用$n_r(u)/n_r$作为从源节点出发的随机游走在第L步走到节点u的概率估计. 具体而言，算法 1 在每次生成随机游走前先定义第 0 步的概率向量$\hat{\boldsymbol{\pi}}^{(0)}=\boldsymbol{e}_s$，即限制第 0 步从节点$s$出发. 对于每一步$l\in[0,L-1]$，首先设置下一步的概率向量$\hat{\boldsymbol{\pi}}^{(\ell+1)}$为零向量，再从当前概率向量$\hat{\boldsymbol{\pi}}^{(\ell)}$中取出有非零概率的节点，从该节点出发采样该节点的邻居，被采样到的邻居即是下一步随机游走的方向. 采样结束后，更新$\hat{\boldsymbol{\pi}}^{(\ell+1)}$中被采样邻居的概率值. 可以注意到，蒙特卡洛算法在生成随机游走时，每步均会调用采样技术从当前节点的出邻居中采样节点. 因此，随机游走各步的采样效率是影响随机游走概率计算效率的关键.

算法 1. 随机游走概率计算的蒙特卡洛方法.

输入：动态有权图$G=(V,E)$，源节点s，步长L，随机游走条数n_r；

输出：$\boldsymbol{\pi}^{(L)}$的估计值.

① for $w=1,2,\cdots,n_r$ do

② 定义 n 维向量$\hat{\boldsymbol{\pi}}^{(0)}$并作初始化：$\hat{\boldsymbol{\pi}}^{(0)}\leftarrow\boldsymbol{e}_s$；

③ for $l=0,1,\cdots,L-1$ do

④ 定义 n 维向量$\hat{\boldsymbol{\pi}}^{(\ell)}$并作初始化：$\hat{\boldsymbol{\pi}}^{(\ell+1)}\leftarrow\boldsymbol{0}$；

⑤ for each $u\in V$ with nonzero $\hat{\pi}^{(\ell)}(u)$ do

⑥ 调用一个采样方法，从集合$N_{\text{out}}(u)$中采样节点并更新向量$\hat{\boldsymbol{\pi}}^{(\ell+1)}$向量对应维；

⑦ end for

⑧ Clear $\hat{\boldsymbol{\pi}}^{(\ell)}$；

⑨ end for

⑩ $\boldsymbol{q}_w \leftarrow \hat{\boldsymbol{\pi}}^{(L)}$；

⑪ end for

⑫ $\hat{\boldsymbol{\pi}}^{(L)} \leftarrow \frac{1}{n_r} \cdot \sum_{w=1}^{n_r} \boldsymbol{q}_w$；

⑬ return $\hat{\boldsymbol{\pi}}^{(L)}$.

2.2 权重采样方法概述

现有工作普遍调用权重采样技术完成随机游走各步的采样. 权重采样技术致力于从一个给定概率分布中采样 1 个元素，具体地，以从图节点u的出邻居集$N_{\text{out}}(u)$中采样出邻居节点为例，应用于随机游走概率计算的权重采样方法会从概率分布$\{A_{uv}/d_u\}$中选取一个出邻居节点v. 此处，A_{uv}/d_u表示节点v的采样概率. 因为$\sum_{v\in N_{\text{out}(u)}} A_{uv}/d_u = 1$，故节点$u$所有出邻居的采样概率形成了一个概率分布，权重采样技术即希望从该概率分布中选出一个节点，使得任意出邻居节点被选中的概率为A_{uv}/d_u. 将权重采样方法运用到算法 1 中实现随机游走概率计算，则每次采样都会有且仅有 1 个邻居被采样，也即每步随机游走都只会走到 1 个节点.

单次权重采样方法的采样时间复杂度和动态更新时间复杂度的理论下界均为$\Omega(1)$，但现有权重采样方法的采样或更新复杂度较$\Omega(1)$差距较大. 这是由于，权重采样的采样结果对于各采样节点是不独立的，具有互斥性. 如果节点v被选中，则说明节点u其它出邻居都未被选中. 这种采样互斥性为权重采样的采样结构设计带来了麻烦，其要求采样结构必须足够复杂方可支持这种采样依赖. 这使得现有的权重采样方法或采样效率不高；或依赖于复杂的采样结构难以在动态图上即时更新. 上述问题成为实现动态有权图上随机游走概率高效计算的阻碍.

下文将对 3 种代表性的权重采样技术及其各自存在的问题作以概述.

2.2.1 别名采样

别名采样（alias sampling）是一种经典的权重采样方法[18-19]，其构建了一个名为别名表（alias table）的采样结构来实现权重采样. 借助别名表，别名采样方法即可在静态有权图上实现$O(1)$时间的权重采样. 但是，由于别名表结构非常复杂，不支持动态图上的实时更新，一旦图结构发生变化（例如出现一条边的增删），别名采样方法都需要花费$O(n_u)$的时间重建别名表，这严重阻碍了别名采样方法在动态有权图上的应用.

2.2.2 前缀和采样

前缀和采样（prefix sum sampling）方法构建了一个二叉搜索树来存储采样概率的前缀和. 借助该二叉搜索树结构，前缀和采样方法可以在动态有权图上以$O\left(1+\log n_u\right)$的时间复杂度完成 1 次采样. 但是，对于动态图上 1 条边的增删，前缀和采样方法需要花费$O(1+\log n_u)$的时间更新该二叉搜索树结构，无法做到实时更新. 特别地，对于各元素采样概率近似相同的情况（无权图上随机游走的各步采样是该情况的一个极端例子），前缀和采样方法仍需要花费$O(\log n_u)$的时间完成 1 次采样，这说明二叉搜索树结构的灵活性较差，无法根据元素的采样概率分布进行采样策略调整，从而造成时间浪费.

2.2.3 拒绝采样

拒绝采样（rejection sampling）[20] 也是一种经典的权重采样方法. 以有权图上节点u处的随机游走采样为例，拒绝采样方法首先从$N_{\text{out}}(u)$中均匀随机地采样一个候选节点v，再以概率A_{uv}/A_u^*接受节点v，以$1-A_{uv}/A_u^*$的概率拒绝该候选节点，其中A_u^*为节点u出边边权的最大值. 拒绝采样方法若拒绝了该候选节点，则继续重复上述过程直至接受了 1 个候选节点. 可以注意到，拒绝采样方法不依赖于任何采样结构，其更新时间消耗仅来自于图结构变化后对值A_u^*的维护，更新时间复杂度为$O\left(1+\log\log n_u\right)$. 但是，拒绝采样完成单次采样的期望时间复杂度为$O\left(1+n_u \cdot A_u^*/d_u\right)$，这一复杂度在边权分布高度偏斜的图上可能接近$O(n_u)$，该复杂度量级较高，会造成不必要的时间浪费.

2.3 硬币翻转采样方法概述

本节将介绍硬币翻转采样定义和现有的硬币翻转采样方法.

硬币翻转采样是一种经典的采样模式，其关注如何以指定概率对各元素进行独立地采样. 具体地，以图节点u处的随机游走采样为例，硬币翻转采样会对节点u的每一个出邻居节点进行独立采样，节点u的每个出邻居节点都将以采样概率被采出. 如果将硬币翻转采样运用到算法 1 中实现随机游走概率计算，那么每次采样可能会得到 0 个或多个邻居，也即每步随机游走可能没有走到任何节点或走到了多个节点. 如果有 1 步随机游走没有走到任何节点，那么这次随机游走模拟就结束在了中间步而没有走到要求的步；如果走到了多个节点，那么下一步就需要从

这些节点出发，分别采样它们的邻居并游走到所有被采样的邻居.

更为形象地，在节点u处的硬币翻转采样过程可以被理解为一个硬币投掷过程，硬币翻转采样在节点u的每个出邻居节点处都独立地抛掷一个硬币. 对于节点u的某一出邻居v，硬币翻转采样在v处抛掷的硬币正面向上的概率为A_{uv}/d_u，反面向上的概率为$1-A_{uv}/d_u$. 硬币翻转采样将返回所有硬币抛掷时正面向上的节点作为采样结果.

对比硬币翻转采样和权重采样，硬币翻转采样的采样结果对各采样元素是独立的：节点u的多个出邻居节点可能被同时采样到. 但是，权重采样的采样结果是互斥的，每次权重采样只会从出邻居集合中采出1个节点. 硬币翻转采样的采样独立性为其采样结构的设计增添了很大的灵活度. 如本文第4节所示，仅需构造一个非常松散的采样结构，现有硬币翻转方法就可以同时做到高效的采样和更新.

2.3.1 硬币翻转采样的研究现状

现有的硬币翻转采样方法大多针对静态采样场景，即元素个数和采样概率始终保持不变的情况，早期Tsai等人[25]提出了时间复杂度为$O(1+n\sqrt{\min\{\bar{p},1-\bar{p}\}})$的算法，之后Bringmann等人[26]将单次采样的时间复杂度改进到最优的$O(1+\mu)$，达到理论下界. Guo等人[21]针对静态影响力最大化问题提出了预先对元素概率排序的方法以获得更好的实际运行效率，相关算法的理论时间复杂度为$O(1+\mu+\log n_u)$.

动态场景下的硬币翻转采样近年来才逐渐受到关注，所述动态场景即指元素数量和采样概率会随时间变化. Yi等人[22]提出了动态硬币翻转采样算法ODSS，同期，Bhattacharya等人[23]提出了SetSampler，这2个方法均同时达到了最优采样时间复杂度$O(1+\mu)$和最优更新时间复杂度$O(1)$.

但是，ODSS和SetSampler算法只能在$O(1)$时间内处理单个元素的变动(增加或删除1个元素，修改1个元素的概率)，而在有权动态图上的随机游走中，节点u的任一边权发生变化或邻居数量发生变化，d_u就会发生变化，从而每个邻居的采样概率A_{uv}/d_u都发生变化，ODSS和SetSampler在这种情况下需要花费$O(n_u)$的时间才能完成一次更新.

2.3.2 BioSampling算法

BioSampling是硬币翻转采样的代表算法，对于含n_u个元素的硬币翻转采样问题，BioSampling算法可以在$O(1+\mu+\log n_u)$的采样时间复杂度下完成1次采样，其中μ指所有元素采样概率之和，$\Omega(1+\mu)$即为该问题采样时间复杂度的理论下界. 值得注意的是，BioSampling算法目前不支持采样元素的动态增删，本文第4节将为其设计支持动态图上实时更新的采样结构.

下文将简述BioSampling算法的具体流程. 以图节点u处的随机游走采样为例，BioSampling算法在读图阶段提前将节点u的出邻居集合划分为$\lceil\log n_u\rceil+1$个组，其中第i组($i\in[1,\lceil\log n_u\rceil]$)包含所有满足$p_v\in\left(\frac{1}{2^i},\frac{1}{2^{i-1}}\right]$的节点$v$，此处$i\in\{1,2,\cdots,\lceil\log n_u\rceil\}$，$p_v=A_{uv}/d_u$指节点$v$的采样概率；第$\lceil\log n_u\rceil+1$组包含所有满足$p_v\in(0,1/2^{\lceil\log n_u\rceil}]$的节点$v$. BioSampling算法依次扫描这$\lceil\log n_u\rceil+1$个组，并在各组中分别进行硬币翻转采样.

在第i组($i\in[1,\lceil\log n_u\rceil]$)的硬币翻转采样中，BioSampling算法首先设置一个采样概率上界$\bar{p}=1/2^{i-1}$，将该采样概率上界$\bar{p}$视为该组内各节点的采样概率并进行硬币翻转采样. 具体地，先产生一个二项分布随机数$b\sim B(n_u,\bar{p})$，再从该组内所有节点中均匀随机地采样c个节点. 对于每个被采样到的节点，称为候选节点v，以$p_v/\bar{p}$的概率采出候选节点v，以$1-p_v/\bar{p}$的概率拒绝采出节点v. 可以注意到，节点v最终被BioSampling算法采出的概率为$\bar{p}\cdot p_v/\bar{p}=p_v$，故BioSampling算法在各组内进行的硬币翻转采样是无偏的.

值得注意的是，在前$\lceil\log n_u\rceil$组中，组内采样概率的上界$\bar{p}$与该组内任一节点v的采样概率p_v最多相差2倍. 因此在期望上，BioSampling算法在该组内采出的候选节点的数量不会超过最终被采样的节点数量的2倍. 因此，BioSampling算法在前$\lceil\log n_u\rceil$组的采样时间开销不超过$1+2\mu+\lceil\log n_u\rceil=O\left(1+\mu+\log n_u\right)$.

对于第$\lceil\log n_u\rceil+1$组的硬币翻转采样，BioSampling算法设置采样概率上界$\bar{p}=1/n_u$并产生二项分布随机数$c\sim B(n_u,\bar{p})$，若$c<1$则直接结束该组内的采样，否则在该组内均匀随机地采样c个候选节点，并以$p_v/\bar{p}$的概率采出候选节点v. 由于$\bar{p}=1/n_u$，故该组采出候选节点的期望数不超过1，不影响BioSampling算法单次采样的时间复杂度.

值得注意的是，上述产生二项分布随机数$c\sim B(n_u,\bar{p})$并作c次均匀随机采样的过程可以通过重复产生几何分布随机数的方法在$O(n_u\bar{p})$的期望时间内实现. 具体地，BioSampling算法首先令$k'=0$，产生一个几何分布随机数$k\sim G(\bar{p})$并采样节点u的第k个邻居作为候选节点；再令$k'=k$，产生一个几何分布随机数$k\sim G(\bar{p})$并采样节点u的第$k+k'$个邻居作为候选节点；重复上述操作直至$k+k'>n_u$.

3 动态图上可实时更新的硬币翻转采样结构

本节提出了一个支持动态更新的硬币翻转采样结构，该结构可以使 BioSampling 算法在 $O(1)$ 时间内处理完动态有权图上 1 条边的增删. 值得注意的是，动态图上的节点增删操作可以转化为多个边增删操作（即逐一增加或删除该节点的所有邻边），边权变化操作可以转化为 1 次边删除操作和 1 次边增加操作. 因此，为了表述方便，本文只关注动态有权图上的边增删操作.

3.1 采样结构

本文在图 1 上各节点处构建了支持实时动态更新的硬币翻转采样结构. 以图上某一节点 u 为例，节点 u 处构建的采样结构包含出邻居节点索引表、出邻居的分组情况表、出邻居分组情况汇总表 3 个部分.

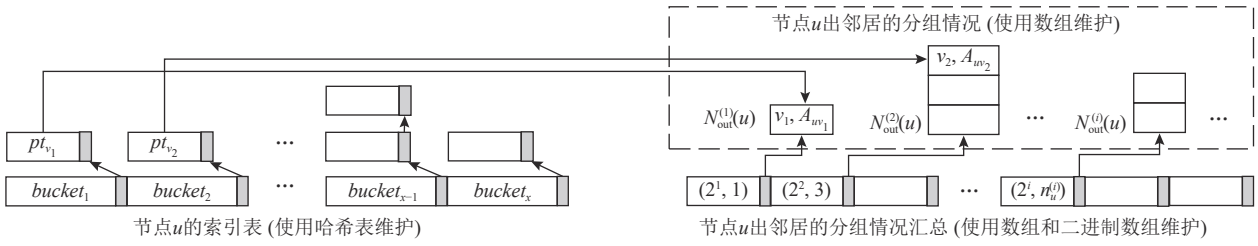


Fig. 1 Sampling structure of coin flip sampling technique

图 1 硬币翻转采样技术的采样结构

1）出邻居节点索引表

出邻居节点索引表（index table）包含节点 u 的所有出邻居的索引. 该索引表使用哈希表维护，哈希表中存储的元素为指向节点 u 各出邻居信息的指针，元素键值（key）为出邻居节点的 ID. 由此，给定节点 u 某一出邻居节点 v 的 ID 信息，借助该索引表即可在 $O(1)$ 时间内找到出邻居节点 v 的相关信息. 值得注意的是，该索引表结构是动态图维护的基本结构. 所有处理动态图的算法都需要建立类似的结构以实时定位到增删的边.

2）出邻居的分组情况表

如第 3.3 节所述，BioSampling 算法会提前将待采样元素按照其采样概率分成多个组，并在各组内分别执行采样操作. 为此，本文为节点 u 的所有出邻居建立分组情况表，该表在形式上包括 $\lceil \log A^*_u \rceil = O\left(\log d_u\right)$ 个组（group），各组均由数组（array）维护，其中第 i 组容纳所有 $A_{uv} \in (2^{i-1}, 2^i]$ 的节点 v 及 A_{uv} 的值，此处 v 指代节点 u 的一个出邻居，$i \in [1, \lceil \log A^*_u \rceil]$. 为了表述的方便，本文用 $N^{(i)}_{out}(u)$ 指代节点 u 分组情况表中的第 i 组. 上述节点 u 的出邻居节点索引表中以 v 为键值的哈希元素，其存储的指针即指向 v 在节点 u 的分组情况表中所在的位置. 由此，给定节点 u 某一出邻居节点 v 的 ID 信息，借助节点 u 的出邻居节点索引表中存储的指针即可在 $O(1)$ 时间内找到节点 v 在其分组情况细节表中存储的位置，进而读取到边 A_{uv} 的权重.

3）出邻居的分组情况汇总表

本文额外为每个图节点 u 建立一个分组情况汇总表，该汇总表由一个数组（array）和一个二进制数组（bit array）维护，用于存储节点 u 分组情况表的一些统计信息. 具体而言，该汇总表的数组结构长度为 $\lceil \log A^*_u \rceil$，该表第 i 项存储 $N^{(i)}_{out}(u)$ 中的元素个数和一个指向 $N^{(i)}_{out}(u)$ 数组结构的指针. 特别地，对于所有非空的 $N^{(i)}_{out}(u)$，其组的 ID（即 i）记录在该汇总表的二维数组结构中，即非空组 ID 对应位置标记为 1，其余组为 0. 由此，借助该汇总表便可完成 BioSampling 中所要求的操作，即将采样元素分组并在各组中依次进行采样. 具体而言，给定节点 u，逐一访问节点 u 分组情况汇总表的二进制数组便可找到所有指向非空 $N^{(i)}_{out}(u)$ 的指针，进而找到节点 u 分组情况表中的所有组，从而得以在各组中完成 BioSampling 算法所要求的采样操作.

3.2 采样结构对图更新操作的处理步骤

本节将具体说明所提采样结构处理动态图上一条边增删操作的方法.

1）边增（edge insertion）操作. 以节点 u 为例，当在图 G 上插入边 (u,v) 时，节点 u 的出邻居数增加. 因此，在边增操作后，该采样结构首先需要在节点 u 的分组情况汇总表中更新出邻居 v 所在组的信息. 为表述方便，此处用 $N^{(i)}_{out}(u)$ 指代 v 所在的组；其次，根据汇总表中存储的指针，找到 $N^{(i)}_{out}(u)$ 对应的数组结构，在数组的最后加入节点 v 及 A_{uv} 的信息. 进一步地，创建一个指向 $N^{(i)}_{out}(u)$ 中节点 v 存储位置的指针，并在节点 u 的邻居节点索引表新增一个哈希元素存储该指针.

2）边删（edge deletion）操作. 以节点 u 为例，当从

图G上删除边(u,v)时，节点u的出邻居数减少. 因此，在边删操作后，该采样结构首先需要在节点u的邻居节点索引表找到其存储的指针，通过该指针找到节点v的相关信息在$N_{\text{out}}^{(i)}(u)$的存储位置，此处用$N_{\text{out}}^{(i)}(u)$指代v所在的组. 然后将v存储在$N_{\text{out}}^{(i)}(u)$中的信息删除，再将$N_{\text{out}}^{(i)}(u)$尾项中的信息补入v在$N_{\text{out}}^{(i)}(u)$中的原位置，保证$N_{\text{out}}^{(i)}(u)$中元素存储的连续性，同时也需要修改尾项元素在邻居节点索引表中的键值，使其指向更新后的位置. 最后，在节点u的分组情况汇总表中更新$N_{\text{out}}^{(i)}(u)$组的信息.

3.3 采样结构的动态更新时间分析

本节将对所提采样结构处理动态图上边增删操作的时间复杂度，以及所提采样结构的空间复杂度进行分析.

引理 1. 在处理图结构边增删的时间开销之外，所提采样结构处理单条边增删操作的时间开销为$O(1)$.

证明. 引理 1 的证明较为直观. 注意，所有处理动态图的算法都需要建立类似邻居节点索引表的结构以实时定位到需要增删的边. 因此，由邻居节点索引表所引起的边增删时间不在本文分析范围之内. 由此，根据第 4.2 节所述更新步骤，所提采样结构处理动态图上边增删的各步操作所消耗的均为$O(1)$. 证毕.

4 动态有权图上的随机游走概率计算方法

本节将硬币翻转采样方法应用至动态有权图上的随机游走概率计算中，提出了一种新颖的高效算法 CoinFlipWalk. CoinFlipWalk 也是一种蒙特卡洛算法，其通过在给定图结构上重复地生成随机游走实现随机游走概率的近似估计. 但与既往方法不同的是，CoinFlipWalk 在每条随机游走中都调用 BioSampling 这一硬币翻转采样方法进行随机游走采样. 下文将证明 CoinFlipWalk 可以在近似最优（忽略 log 因子）的时间复杂度内完成定义 1 中所要求的随机游走概率近似计算. 结合第 4 节提出的支持实时动态更新的硬币翻转采样结构，CoinFlipWalk 即可同时做到近似最优的随机游走概率计算复杂度和最优的采样结构更新复杂度.

算法 2. CoinFlipWalk 的单步采样操作.

输入：满足$\hat{\pi}^{(\ell)}(u)>0$的节点u；

输出：更新后$\boldsymbol{\pi}^{(\ell+1)}$的估计值.

① for each non-empty group $N_{\text{out}}^{(i)}(u)$ of u do

② Let $p^*=\dfrac{\hat{\pi}^{(\ell)}(u)\cdot 2^i}{d_u}$ for all $v\in N_{\text{out}}^{(i)}(u)$;

③ if $p^*\geqslant 1$ then

④ for each $v\in N_{\text{out}}^{(i)}(u)$ do

⑤ $\hat{\pi}^{(\ell+1)}(v)\leftarrow\hat{\pi}^{(\ell+1)}(v)+\dfrac{\hat{\pi}^{(\ell)}(u)\cdot A_{uv}}{d_u}$;

⑥ end for

⑦ else

⑧ 定义$n_u^{(i)}$以记录集合 in $N_{\text{out}}^{(i)}(u)$中的节点数；

⑨ $idx=0$;

⑩ while $idx<n_u^{(i)}$ do

⑪ 生成一个几何分布随机数 $rg\sim G(p^*)$;

⑫ $idx\leftarrow idx+rg$;

⑬ 定义 v 以表示集合 $N_{\text{out}}^{(i)}(u)$中的第idx个节点数；

⑭ 生成一个均匀随机数 $r\sim U(0,1)$;

⑮ if $r<\dfrac{A_{uv}}{2^i}$ then $\hat{\pi}^{(\ell+1)}(v)\leftarrow\hat{\pi}^{(\ell+1)}(v)+1$;

⑯ end if

⑰ end while

⑱ end if

⑲ end for

⑳ return $\hat{\boldsymbol{\pi}}^{(\ell+1)}$.

4.1 CoinFlipWalk 算法说明

CoinFlipWalk 算法的主体框架仍为蒙特卡洛方法框架，该框架已在算法 1 中展示. CoinFlipWalk 中使用的硬币翻转采样方法具体如算法 2 所示，该采样方法以 BioSampling 为基础，且针对随机游走概率计算做了适应性调整. 将算法 2 嵌入算法 1 的行⑥即得到完整的 CoinFlipWalk 算法.

具体而言，给定动态权重图$G=(V,E)$、源节点s、随机游走步长L和随机游走条数n_r，CoinFlipWalk 算法从源节点s处生成n_r条“广义”的随机游走，其中各条随机游走在其游走第ℓ步的结果都由一个n维向量$\hat{\boldsymbol{\pi}}^{(\ell)}$记录，$\ell\in\{0,1,\cdots\}$. $\hat{\boldsymbol{\pi}}^{(0)}$被初始化为$\boldsymbol{e}_s$，其余所有$\ell\geqslant 1$的$\hat{\boldsymbol{\pi}}^{(0)}$向量均被初始化为$n$维的全 0 向量. 算法 2 展示了 CoinFlipWalk 中“广义”随机游走第ℓ步的生成过程，其本质上是基于向量$\hat{\boldsymbol{\pi}}^{(\ell)}$的值对向量$\hat{\boldsymbol{\pi}}^{(\ell+1)}$的更新过程. 具体说明为：

对于图上各节点u，若$\hat{\pi}^{(\ell)}(u)$非零，则依据下述步骤使用$\hat{\pi}^{(\ell)}(u)$更新节点u的出邻居节点v所对应的$\hat{\pi}^{(\ell+1)}(v)$.

将节点u的所有出邻居节点分成多个组，其中第i组容纳所有$A_{uv}\in(2^{i-1},2^i]$的出邻居节点v；设第i组中所有节点的采样概率为$p^*=\left(\hat{\pi}^{(\ell)}(u)\cdot 2^i/d_u\right)$. 若$p^*\geqslant 1$，

则更新第i组中所有节点v所对应的$\hat{\pi}^{(\ell+1)}(v)$：

$$\hat{\pi}^{(\ell+1)}(v) \leftarrow \hat{\pi}^{(\ell+1)}(v) + \frac{\hat{\pi}^{(\ell)}(u)\cdot A_{uv}}{d_u}.$$

反之，若$p^* < 1$，则从第i组中进行硬币翻转采样，对应算法2中的行⑧~⑭. 首先设定第i组中每个节点v的采样概率均为$\hat{\pi}^{(\ell)}(u)\cdot 2^i/d_u$来采样候选节点. 因为此时各节点的采样概率相同，故第i组中的硬币翻转采样退化为二项分布采样，如第3.3节所述，二项分布采样可以用等价的几何分布采样代替，即从第i组中的第1个节点开始，产生几何分布随机数$k \sim G\left(p^*\right)$，其中$p^* = \hat{\pi}^{(\ell)}(u)\cdot 2^i/d_u$，返回第$k$个邻居作为第一个候选节点. 之后，继续从第$k+1$个邻居开始并重复上述操作，直至跳过第$i$组中的所有节点. 对于所有采样到的候选节点，需要保证其采样概率正确，所以还需执行算法2中的行⑫~⑭. 对每个采样到的候选节点v，以$A_{uv}/2^i$的概率对节点v所对应的$\hat{\pi}_w^{(\ell+1)}(v)$进行更新：

$$\hat{\pi}^{(\ell+1)}(v) \leftarrow \hat{\pi}^{(\ell+1)}(v) + 1,$$

以$1-A_{uv}/2^i$的概率不更新$\hat{\pi}^{(\ell+1)}(v)$. 至此，CoinFlipWalk算法的一步采样结束，完成了基于向量$\hat{\boldsymbol{\pi}}^{(\ell)}$的值对向量$\hat{\boldsymbol{\pi}}^{(\ell+1)}$的更新过程. 根据算法1的框架，从$\ell = 0$开始重新上述过程$L$次，即可完成一次随机游走模拟对概率向量$\hat{\boldsymbol{\pi}}^{(L)}$的更新. 重复模拟$n_r$次，即可得到4.2节所分析的有理论保证的估计结果.

4.2 理论分析

本节将对CoinFlipWalk算法的理论性质进行分析并给出详细证明.

4.2.1 无偏性

本节首先证明CoinFlipWalk算法计算得到的随机游走概率是无偏的. 具体如引理1所示.

引理1. 对于$\forall u \in V$、$\forall \ell \in [0, L]$，将算法1的行⑥替换为算法2后，执行算法1所返回的$\hat{\pi}^{(\ell)}$满足：

$$\mathbb{E}\left[\hat{\pi}^{(\ell)}(u)\right] = \pi^{(\ell)}(u).$$

证明. 引理1可通过数学归纳法证明. 首先，考虑起始状态：$\hat{\boldsymbol{\pi}}^{(0)} = \boldsymbol{e}_s$，故对于$\forall u \in V$，有：$\mathbb{E}\left[\hat{\pi}^{(0)}(u)\right] = \pi^{(0)}(u)$. 进一步地，假设对某$\ell \in [0, L-1]$，$\mathbb{E}\left[\hat{\pi}^{(\ell)}(u)\right] = \pi^{(\ell)}(u)$对$\forall u \in V$成立，由于：

$$\mathbb{E}\left[\hat{\pi}^{(\ell+1)}(v) \mid \hat{\boldsymbol{\pi}}^{(\ell)}\right] = \sum_{u\in N_{\text{in}}(v)} \frac{A_{uv}\cdot \hat{\pi}^{(\ell)}(u)}{d_u}. \tag{4}$$

因此，$\mathbb{E}\left[\hat{\pi}^{(\ell+1)}(v)\right] = \mathbb{E}\left[\mathbb{E}\left[\hat{\pi}^{(\ell+1)}(v) \mid \hat{\boldsymbol{\pi}}^{(\ell)}\right]\right]$. 进而有，

$$\mathbb{E}\left[\hat{\pi}^{(\ell+1)}(v)\right] = \sum_{u\in N_{\text{in}}(v)} \frac{\boldsymbol{A}_{uv}\cdot \mathbb{E}\left[\hat{\pi}^{(\ell)}(u)\right]}{d_u}.$$

代入$\mathbb{E}\left[\hat{\pi}^{(\ell)}(u)\right] = \pi^{(\ell)}(u)$的假设和式(2)，可得：

$$\mathbb{E}\left[\hat{\pi}^{(\ell+1)}(v)\right] = \sum_{u\in N_{\text{in}}(v)} \frac{\boldsymbol{A}_{uv}\cdot \pi^{(\ell)}(u)}{d_u} = \pi^{(\ell+1)}(v).$$

下文将具体说明式(4)成立的原因. 回顾算法2，对于第i组中所有节点v，若$\left(\hat{\pi}^{(\ell)}(u)2^i/d_u\right) \geqslant 1$，则算法1将$\hat{\pi}^{(\ell+1)}(v)$确定地增加$\hat{\pi}^{(\ell)}(u)\cdot A_{uv}/d_u$. 若$\left(\hat{\pi}^{(\ell)}(u)\cdot 2^i/d_u\right) < 1$，则算法1以$\hat{\pi}^{(\ell)}(u)\cdot 2^i/d_u$的采样概率从第$i$组中采样出节点$v$，再以$A_{uv}/2^i$的概率将$\hat{\pi}^{(\ell+1)}(v)$增加1. 因此，当$\left(\hat{\pi}^{(\ell)}(u)\cdot 2^i/d_u\right) < 1$时，$\hat{\pi}^{(\ell+1)}(v)$的期望增加值仍为$\hat{\pi}^{(\ell)}(u)\cdot\frac{2^i}{d_u}\cdot\frac{A_{uv}}{2^i} = \hat{\pi}^{(\ell)}(u)\cdot\frac{A_{uv}}{d_u}$，故式(4)可得，从而引理1得证. 证毕.

4.2.2 单次采样时间

本节进一步分析CoinFlipWalk完成单次硬币翻转采样的时间开销.

引理2. 对于$\forall u \in V$、$\forall \ell \in [0, L-1]$，算法2的期望时间复杂度上界为$O\left(1+\log n_u\right)$.

证明. 令$Cost^{(\ell)}(u)$表示在节点u处执行1次算法2的期望时间复杂度，令$Cost^{(\ell)}(u,v)$表示算法2中更新$\hat{\pi}^{(\ell+1)}(v)$所花费的时间，根据算法2有：

$$Cost^{(\ell)}(u) \leqslant \log A_u^* + \sum_{v\in N_{\text{out}(u)}} Cost^{(\ell)}(u,v), \tag{5}$$

其中，$\log A_u^*$项与本文第4节所提采样结构有关，即要完成节点u处的硬币翻转采样，需要至少花费$O(1)$时间遍历节点u分组情况细节表中的每个非空组，所以算法2最多需要花费和组数相同的时间遍历非空组，故算法2的时间复杂度中包含$O\left(\log A_u^*\right)$. 式(5)中$\sum_{v\in N_{\text{out}(u)}} Cost^{(\ell)}(u,v)$表示算法2更新节点$u$的所有邻居节点$v$的$\hat{\pi}^{(\ell+1)}(v)$所花费的时间，该时间即为算法2在各个非空组中的采样时间之和.

值得注意的是式(5)中的$\log A_u^*$项可被进一步减小为$\log n_u$. 具体而言，设第i组中所有节点的采样概率为$\hat{\pi}^{(\ell)}(u)\cdot 2^i/d_u$，算法2在扫描节点$u$的分组情况细节表时，可以$1-1/n_u$的概率跳过所有$2^i/d_u$小于$1/n_u^2$的组. 这是因为，节点$u$最多有$n_u$个邻居满足$2^i/d_u \leqslant 1/n_u^2$，这些邻居中至少有1个被采样到的概率不超过$1-\left(1-\frac{1}{n_u^2}\right)^{n_u}$，进而不超过$\frac{1}{n_u}$，也即有大于$1-1/n_u$的概率是不需要采样所有这些满足条件的邻居节点的. 所以，算法2在运行之初可以产生一个$[0,1]$之间的均匀随机数r，若$r \leqslant 1-1/n_u$，则跳过所有满足$\frac{2^i}{d_u} \leqslant 1/n_u^2$的分组；否则逐一扫描这些分组中的邻居节点以判断是否需要采样，该过程最多消耗$O(n_u)$的时间. 注意，满足$\frac{2^i}{d_u} \geqslant 1/n_u^2$的分组最多有$2\log n_u$组. 因此，$\log A_u^*$

项从期望上可被减小为: $2\log n_u + n_u \cdot \frac{1}{n_u} = O(1+\log n_u)$. 实现时, 可以始终维护首个采样概率下界超过$1/n_u^2$的组 ID, 记为$j$, 若产生的随机数$r \leqslant 1-1/n_u$, 直接从第$j$组开始扫描.

另一方面, 根据算法 2, 若$\hat{\pi}^{(\ell)}(u)2^i/d_u \geqslant 1$, 则有$Cost^{(\ell)}(u,v)=1 \leqslant \left(\hat{\pi}^{(\ell)}(u)2^i/d_u\right)$; 若 $\left(\hat{\pi}^{(\ell)}(u)\cdot 2^i/d_u\right)<1$, 则以$\hat{\pi}^{(\ell)}(u)\cdot 2^i/d_u$的概率有$Cost^{(\ell)}(u,v)=1$. 因此,

$$\mathbb{E}\left[Cost^{(\ell)}(u,v)\mid \hat{\boldsymbol{\pi}}^{(\ell)}\right] \leqslant \hat{\pi}^{(\ell)}(u)\frac{2^i}{d_u} \leqslant \hat{\pi}^{(\ell)}(u)\cdot\frac{2A_{uv}}{d_u}.$$

综上, $Cost^{(\ell)}(u)$的条件期望可被表示为:

$$\mathbb{E}\left[Cost^{(\ell)}(u)\mid \hat{\boldsymbol{\pi}}^{(\ell)}\right] \leqslant \log n_u + \sum_{v\in N_{\text{out}(u)}} \hat{\pi}^{(\ell)}(u)\frac{2A_{uv}}{d_u}.$$

由于$\mathbb{E}\left[Cost^{(\ell)}(u)\right]=\mathbb{E}\left[\mathbb{E}\left[Cost^{(\ell)}(u)\mid \hat{\boldsymbol{\pi}}^{(\ell)}\right]\right]$成立, 结合$Cost^{(\ell)}(u)$的条件期望, 并由引理 1 可进一步得到式(5)的上界:

$$\mathbb{E}\left[Cost^{(\ell)}(u)\right] \leqslant \log n_u + \sum_{v\in N_{\text{out}(u)}} \pi^{(\ell)}(u)\cdot\frac{2A_{uv}}{d_u}.$$

注意 $\sum_{v\in N_{\text{out}(u)}} \pi^{(\ell)}(u)\cdot\frac{2A_{uv}}{d_u} \leqslant \sum_{v\in N_{\text{out}(u)}} \pi^{(\ell+1)}(v) \leqslant 1$始终成立, 故引理 2 得证. 证毕.

4.2.3 CoinFlipWalk 时间复杂度

最后, 本节证明 CoinFlipWalk 算法可以在近似最优的时间复杂度下完成定义 1 中所要求的随机游走概率计算问题. 定理 3 给出了满足定义 1 所需的随机游走次数. 结合定理 3 和引理 2 可得 CoinFlipWalk 的计算复杂度为$\tilde{O}(L^2/\delta)$. 此处$\tilde{O}$为省略 log 项的大 O 表示法. 值得注意的是, 该问题的平凡下界为$\Omega(1/\delta)$, L常被设置为常数. 因此, 如果忽略 log 系数, CoinFlipWalk 算法的期望时间复杂度达到了理论下界. 以下为引理 3 及其证明.

引理 3. 当$n_r \geqslant \frac{L}{p_f\cdot c^2\delta}$时, CoinFlipWalk 所返回的$\hat{\boldsymbol{\pi}}^{(L)}$对于任意节点$u\in V$以不小于$1-p_f$的概率满足:

$$|\pi^{(L)}(u)-\hat{\pi}^{(L)}(u)| \leqslant c\cdot\max\left\{\pi^{(L)}(u),\delta\right\}.$$

证明. 为证明引理 3, 先对$\forall u\in V$求出$\hat{\pi}^{(L)}(u)$方差的上界, 再借助切比雪夫不等式得到随机游走条数n_r的上界.

回顾算法 2, 对于采样概率小于 1 的组i内的邻居v, $\hat{\pi}^{(\ell+1)}(v)$以$\hat{\pi}^{(\ell)}(u)\cdot A_{uv}/d_u$的概率增加 1, 否则不增加. 令$r^{(\ell)}{}_u(v)$为算法 2 在节点$u$处进行硬币翻转采样时$\hat{\pi}^{(\ell+1)}(v)$的增量, 则有:

$$Var\left[r_u^{(\ell)}(v)\mid \hat{\boldsymbol{\pi}}^{(\ell)}\right] \leqslant \mathbb{E}\left[\left(r_u^{(\ell)}(v)\right)^2\mid \hat{\boldsymbol{\pi}}^{(\ell)}\right],$$

进而有,

$$Var\left[r_u^{(\ell)}(v)\mid \hat{\boldsymbol{\pi}}^{(\ell)}\right] \leqslant 1\cdot\frac{A_{uv}\cdot\hat{\pi}^{(\ell)}(u)}{d_u}.$$

因为$\hat{\pi}^{(\ell+1)}(v)=\sum_{u\in N_{\text{in}}(v)} r_u^{(\ell)}(v)$, 以及硬币翻转采样的独立性, 有:

$$Var\left[\hat{\pi}^{(\ell+1)}(v)\mid \hat{\boldsymbol{\pi}}^{(\ell)}\right]=\sum_{u\in N_{\text{in}}(v)} Var\left[r_u^{(\ell)}(v)\mid \hat{\boldsymbol{\pi}}^{(\ell)}\right].$$

进而有:

$$Var\left[\hat{\pi}^{(\ell+1)}(v)\mid \hat{\boldsymbol{\pi}}^{(\ell)}\right] \leqslant \sum_{u\in N_{\text{in}}(v)} \frac{A_{uv}\cdot\hat{\pi}^{(\ell)}(u)}{d_u}.$$

进一步地, 由于方差满足全方差公式(total variance law), 故有:

$$\begin{aligned}Var\left[\hat{\pi}^{(L)}(u)\right] =& \mathbb{E}\left[Var\left[\hat{\pi}^{(L)}(u)\mid\hat{\pi}^{(L-1)}\right]\right]+\\ & Var\left[\mathbb{E}\left[\hat{\pi}^{(L)}(u)\mid\hat{\pi}^{(L-1)}\right]\right].\end{aligned}$$

因此, $Var\left[\hat{\pi}^{(L)}(u)\right]$可被表示为: $\mathbb{E}\left[Var\left[\hat{\pi}^{(L)}(u)\mid\hat{\pi}^{(L-1)}\right]\right]+Var\left[\sum_{x\in N_{\text{in}}(u)}\frac{A_{xu}\cdot\hat{\pi}^{(L-1)}(x)}{d_x}\right]$. 注意$\frac{A_{xu}}{d_x}=\hat{\pi}_x^{(1)}(u)$, 所以$Var\left[\sum_{x\in N_{\text{in}}(u)}\frac{A_{xu}\cdot\hat{\pi}^{(L-1)}(x)}{d_x}\right]=Var\left[\sum_{x\in N_{\text{in}}(u)}\pi_x^{(1)}(u)\cdot\hat{\pi}^{(L-1)}(x)\right]$, 这也可以写作$Var\left[\sum_{x\in V}\pi_x^{(1)}(u)\cdot\hat{\pi}^{(L-1)}(x)\right]$. 根据全方差公式, 上述推导可以得到更一般的结论: 对于$\ell\in[1,L]$, 有

$$\begin{aligned}&Var\left[\sum_{u\in V}\pi^{(L-\ell)}(x,u)\cdot\hat{\pi}^{(\ell)}(x)\right]=\\ &\mathbb{E}\left[Var\left[\sum_{x\in V}\pi^{(L-\ell)}(x,u)\cdot\hat{\pi}^{(\ell)}(x)\mid\hat{\pi}^{(\ell-1)}\right]\right]+\\ &Var\left[\sum_{y\in V}\pi^{(L-\ell+1)}(y,u)\cdot\hat{\pi}^{(\ell-1)}(y)\right].\end{aligned}\tag{6}$$

由式(6)推导可知, 调用一次全方差公式, 可得:

$$\begin{aligned}Var\left[\hat{\pi}^{(L)}(u)\right] =& \mathbb{E}\left[Var\left[\hat{\pi}^{(L)}(u)\mid\hat{\boldsymbol{\pi}}^{(L-1)}\right]\right]+\\ & Var\left[\mathbb{E}\left[\hat{\pi}^{(L)}(u)\mid\hat{\boldsymbol{\pi}}^{(L-1)}\right]\right]=\\ & \mathbb{E}\left[Var\left[\hat{\pi}^{(L)}(u)\mid\hat{\boldsymbol{\pi}}^{(L-1)}\right]\right]+\\ & Var\left[\sum_{x\in V}\pi_x^{(1)}(u)\cdot\hat{\pi}^{(L-1)}(x)\right],\end{aligned}$$

反复调用式(6), $Var\left[\hat{\pi}^{(L)}(u)\right]$最终可被写为:

$$Var\left[\hat{\pi}^{(L)}(u)\right]=\sum_{\ell=1}^{L}\mathbb{E}\left[Var\left[\sum_{u\in V}\hat{\pi}^{(\ell)}(u)\cdot\pi^{(L-\ell)}(u,v)\mid\hat{\boldsymbol{\pi}}^{(\ell-1)}\right]\right].$$

注意, 对于$\forall u\in V$, $\hat{\pi}^{(\ell)}(u)$的取值在给定$\hat{\boldsymbol{\pi}}^{(\ell-1)}$后是独立的. 因此,

$$\begin{aligned}&Var\left[\sum_{x\in V}\pi^{(L-\ell)}(x,u)\cdot\hat{\pi}^{(\ell)}(x)\mid\hat{\boldsymbol{\pi}}^{(\ell-1)}\right]=\\ &\sum_{x\in V}\left(\pi^{(L-\ell)}(x,u)\right)^2\cdot Var\left[\hat{\pi}^{(\ell)}(x)\mid\hat{\boldsymbol{\pi}}^{(\ell-1)}\right].\end{aligned}\tag{7}$$

$$Var\left[\hat{\pi}^{(\ell)}(x)\,|\,\hat{\boldsymbol{\pi}}^{(\ell-1)}\right]=\sum_{y\in N_{\text{in}}(x)}\frac{A_{yx}\cdot\hat{\pi}^{(\ell-1)}(y)}{d_y}. \tag{8}$$

将式(8)代入式(7)即可得到：

$$Var\left[\sum_{x\in V}\pi^{(L-\ell)}(x,u)\cdot\hat{\pi}^{(\ell)}(x)\,|\,\hat{\boldsymbol{\pi}}^{(\ell-1)}\right]\leqslant\sum_{y\in V}\pi^{(L-\ell+1)}(y,u)\cdot\hat{\pi}^{(\ell-1)}(y).$$

由此，$Var\left[\hat{\pi}^{(L)}(u)\right]$可被写为：

$$Var\left[\hat{\pi}^{(L)}(u)\right]\leqslant\sum_{\ell=1}^{L}\sum_{y\in V}\pi^{(L-\ell+1)}(y,u)\cdot\mathbb{E}\left[\hat{\pi}^{(\ell-1)}(y)\right],$$

代入 $\hat{\pi}^{(\ell-1)}(y)$ 的无偏性，即有：

$$Var\left[\hat{\pi}^{(L)}(u)\right]\leqslant\sum_{\ell=1}^{L}\pi^{(L)}(u)=L\cdot\pi^{(L)}(u).$$

由此得到$\hat{\pi}^{(L)}(u)$方差的上界.

进一步地，将该方差界代入切比雪夫不等式(Chebyshev's inequality)中：

$$Pr\left\{\left|\pi^{(L)}(u)-\hat{\pi}^{(L)}(u)\right|\geqslant c\cdot\pi^{(L)}(u)\right\}\leqslant\frac{Var\left[\hat{\pi}^{(L)}(u)\right]}{c^2\left(\pi^{(L)}(u)\right)^2},$$

并将其代入所得方差界，即有：

$$Pr\left\{\left|\pi^{(L)}(u)-\hat{\pi}^{(L)}(u)\right|\geqslant c\cdot\pi^{(L)}(u)\right\}\leqslant\frac{L}{n_r c^2\cdot\pi^{(L)}(u)}.$$

因此，当$n_r\geqslant\dfrac{L}{p_f\cdot\delta\cdot c^2}$，CoinFlipWalk 计算得到的随机游走概率近似向量$\hat{\boldsymbol{\pi}}^{(L)}$满足定义 1 所示的误差要求. 本引理由此得证. 证毕.

5 实验结果分析

本节展示 CoinFlipWalk 在大规模真实图和合成图上的实验结果. 实验均用 C++实现，使用 g++编译器且开启了 O3 优化进行编译. 本节所涉实验均在一台配备有 Intel 2.20 GHz CPU 和 754 GB 内存的 Linux 机器上完成.

5.1 实验设置

5.1.1 基线方法

本文将 CoinFlipWalk 与 2 种基线方法 PrefixWalk 和 RejectionWalk 进行了比较，这 2 种方法也是蒙特卡洛方法，其在随机游走各步分别调用前缀和采样方法以及拒绝采样方法. 具体而言，CoinFlipWalk 算法在算法 1 的行⑥调用算法 2，而 PrefixWalk 和 RejectionWalk 算法在算法 1 的行⑥分别调用了前缀和采样方法以及拒绝方法. 由于别名采样方法仅支持静态加权图且每次边插入/删除操作都必须完全重建索引结构，因此本文在实验中未包括别名方法. 此外，本文采用幂方法计算有权图上随机游走概率的真值. 幂方法初始化$\boldsymbol{\pi}^{(0)}=\boldsymbol{e}_s$，并重复计算$\boldsymbol{\pi}^{(i+1)}\leftarrow\boldsymbol{P}\boldsymbol{\pi}^{(i)}$，直到得出$\boldsymbol{\pi}^{(L)}$. 在本文中，$L$为常数整数，在实验中除非另有说明，设$L$=10.

5.1.2 数据集

本文在 4 个真实有权图和 2 个合成图上进行了实验. 为了方便参考，表 2 总结了 6 个真实图数据集的统计信息. 其中，$\cos^2\varphi$ 反映有权图的不平衡性，$\cos^2\varphi$ 的值越小，图权重分布的不平衡性越大. 真实图分别为：Tags (TAG)[27], Threads (TH)[27], Reddit (RR)[28-29] 和 Colisten (CS)[30]. 数据集 TAG 和 TH 都源自问答网站 Stack Overflow. 具体来说，TAG 和 TH 数据集中的节点代表问题标签，边权表示 2 个标签在同一个问题系列中同时出现的频率. 数据集 RR 是根据 Reddit 网站上的评论构建出的网络，其中的节点代表用户，边的权重代表用户评论用户的帖子的次数. 数据集 CS 是一个歌曲数据集，歌曲与图节点对应，边权是 2 首歌曲出现相似音乐片段的次数. 合成图包括了 2 个基于子图模式(motif)生成权重的有权图，即 IndoChina (IC)[31-32] 和 Twitter (TW)[33-34]. 这 2 个数据集本身是真实世界中的无权图. 本文参考已有文献 [35-36] 将这 2 个数据集转化为基于子图模式的有权图，对于每一条边，设置其参与的特定子图模式，如一个三角形子图的数量作为权重.

Table 2 Parameters of Datasets

表 2 数据集参数

数据集	节点数n	边数m	$\frac{m}{n}$	$\cos^2\varphi$
TAG	49 945	8 294 604	166	0.27
RR	8 396 162	870 580 840	103	0.90
TH	2 321 767	42 012 344	18	0.97
CS	3 604 308	3 854 964 026	1 070	0.29
IC	7 414 768	295 191 370	40	0.31
TW	41 652 230	1 687 583 684	40	0.34

5.1.3 测试方法和评价指标

本文根据n_u/m的分布随机生成 10 个查询节点. 所有实验结果均是每种方法以随机生成的 10 个查询节点为源节点进行随机游走的平均结果. 对于更新时间的测试，我们从原始图中随机删除 100 000 条边，然后将这 100 000 条边重新插入图中. 对于每种方法，我们计算 200 000 次边更新操作中的平均更新时间开销作为该方法的更新开销. 对于真实图上各方法随机游走的近似误差，我们的评价指标包括 Precision@50、*MaxError*. 其中，Precision@k 是衡量近似精度的

一个流行指标，定义为这些近似方法返回的结果中概率估计值最大的前k个节点与通过基准真值得到的实际概率最大的前k个节点相符的百分比. 实验时，可以通过调整每种方法中的误差参数δ值来达到所需的 Precision@k. δ值越小，现有方法模拟的随机游走次数就越多. 因此δ值越小，每种方法在随机游走模拟过程中花费的查询时间就越长，达到的 Precision@k值也就越高. 本文设置$k=50$. *MaxError* 是另一个常用的衡量指标，该指标量化了所求的近似结果与真值相比的近似误差，定义为$MaxError=\max\limits_{u\in V}\left|\pi^{(L)}(u)-\hat{\pi}^{(L)}(u)\right|$.

5.2 查询时间与更新时间开销的相关分析

本节对各方法在动态有权图上计算和更新随机游走概率的相对大小关系进行了对比，图 2 具体展示了 CoinFlipWalk 及其他方法在达到相近 Precision@50 结果时的随机游走概率计算时间(查询时间)和动态更新时间，各方法名旁显示的数字为各方法所达到的 Precision@50 结果. 根据图 2 所示结果，PrefixWalk 算法在各数据集上的随机游走概率动态更新时间普遍较高，RejectionWalk 算法的随机游走概率动态更新时间较低但计算时间较高，而本文所提的 CoinFlipWalk 算法的随机游走概率计算和更新时间均较低. 特别地，由此体现出 CoinFlipWalk 的优越性.

5.3 查询时间与近似精度的相关分析

本节对各方法的随机游走概率计算效率进行了对比，图 3 具体展示了真实有权图上 CoinFlipWalk 及其他算法在随机游走概率计算中的 Precision@50 与查询时间的变化关系，图中各算法对应查询时间从左到右依次对应取 0.1, 0.01, …, 10^{-5}，查询时间越小，各算法所产生的随机数量越多，算法运行时间越长. 值得注意的是，相比于 PrefixWalk 和 RejectionWalk，CoinFlipWalk 在各数据集上都可以在更短的计算时间内达到更高的 Precision@50，这体现出 CoinFlipWalk 的优越性. 此外，图 4 展示了真实有权图上 CoinFlipWalk 及其他方法在随机游走概率计算中所得结果的 *MaxError* 与查询时间的变化关系. 类似地，相比于 PrefixWalk 和 RejectionWalk，CoinFlipWalk 在各数据集上都可以在更短的查询时间内达到更小的 *MaxError*，这再次体现出 CoinFlipWalk 的计算高效性. 另外值得注意的是，从图 3 和图 4 中可以看出，PrefixWalk 在边权分布不均(即 $\cos^2\varphi$较小的图数据集)的图上较在边权分布

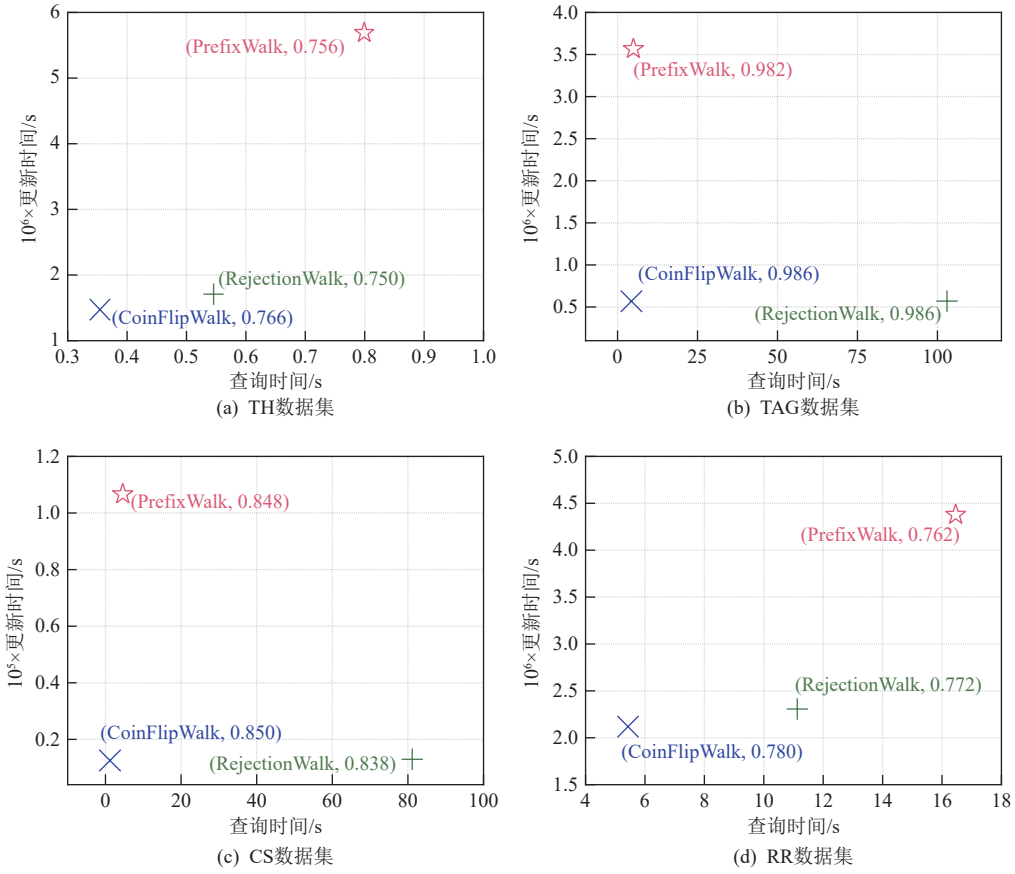


Fig. 2 Update time varies with query time on real-world weighted graphs in different methods

图 2 在真实有权图上各方法的更新时间随查询时间的变化

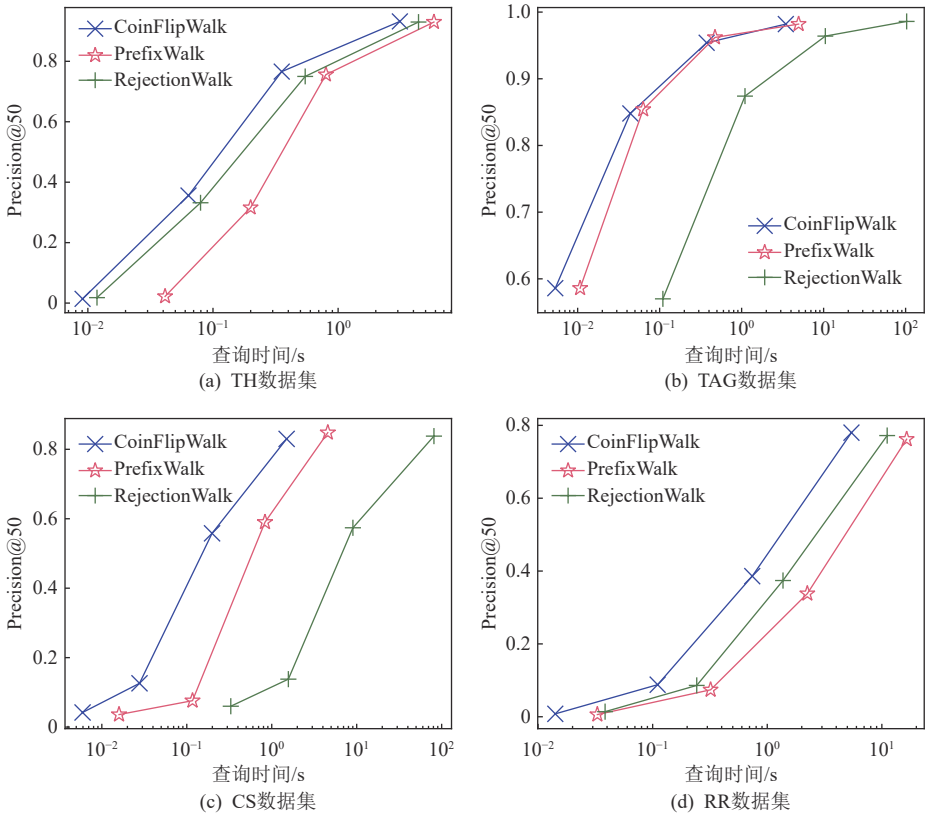


Fig. 3 Presicion@50 of different methods varies with query time on real-world weighted graphs

图 3 在真实有权图上各方法的 Presicion@50 随查询时间的变化

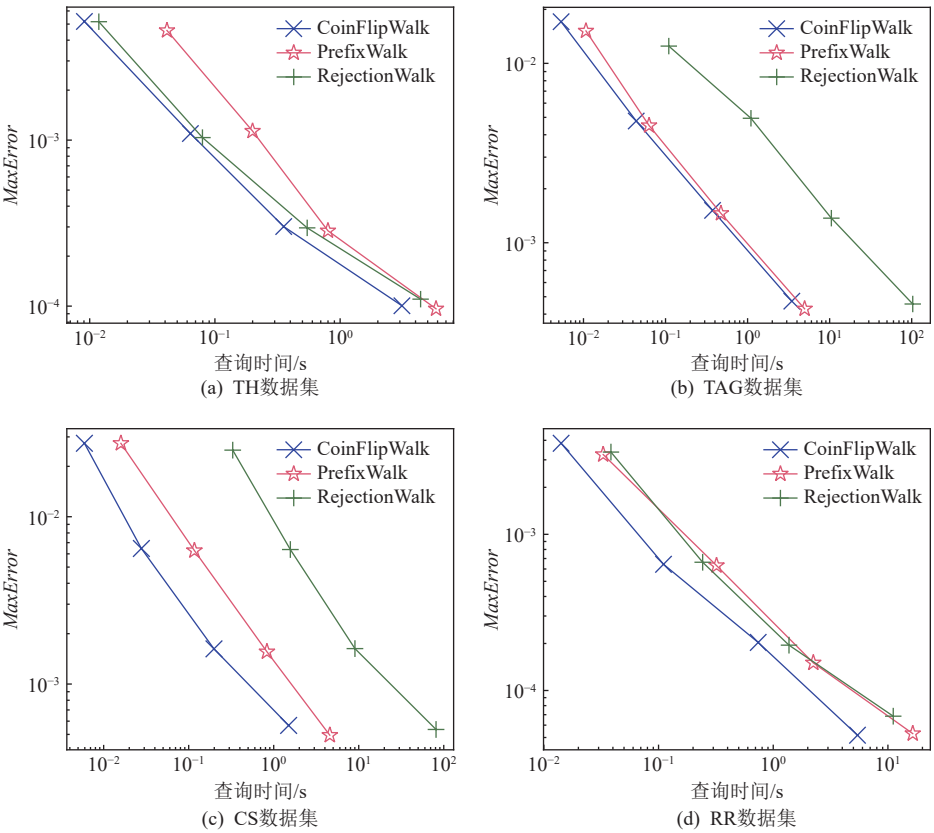


Fig. 4 *MaxError* of different methods varies with query time on real-world weighted graphs

图 4 在真实有权图上各方法的 *MaxError* 随查询时间的变化

较为均匀(即 $\cos^2\varphi$较大的图数据集)的图上的计算效率更高.与之相对地, RejectionWalk 在边权分布均匀的图上较在边权分布较为不均的图上的计算效率更高.进一步地, 本节还对 CoinFlipWalk 及其他算法在合成图上的随机游走计算效率进行了对比. 其中, 图 5 和图 6 分别展示了 CoinFlipWalk, PrefixWalk 和 RejectionWalk 算法在 2 个合成图数据集上进行随机游走概率计算时所得 Precision@50 和 *MaxError* 与查询时间的变化关系.与前文所述类似, CoinFlipWalk 在各数据集上都可以在更短的计算时间内达到更大的 Precision@50 和更小的 *MaxError*.进一步地, 本文还分别将 CoinFlipWalk, PrefixWalk 和 RejectionWalk 算法计算所得的随机游走概率近似结果应用于局部社区发现(local clustering)的场景中, 按照 sweep 方法[35] 基于随机游走概率近似结果发现给定源节点周围的社区, 并计算所得社区的导度(一种社区质量衡量指标, 导度值越低, 所得社区质量越好). 图 7 展示了 CoinFlipWalk, PrefixWalk 和 RejectionWalk 三个算法所得社区的导度与算法计算随机游走概率时间之间的相对变化关系, 可以看出, CoinFlipWalk 可以在各数据集上用更短的查询时间取得更小的导度, 这再次证明 CoinFlipWalk 的高效性, 也体现出 CoinFlipWalk 的实际应用价值.

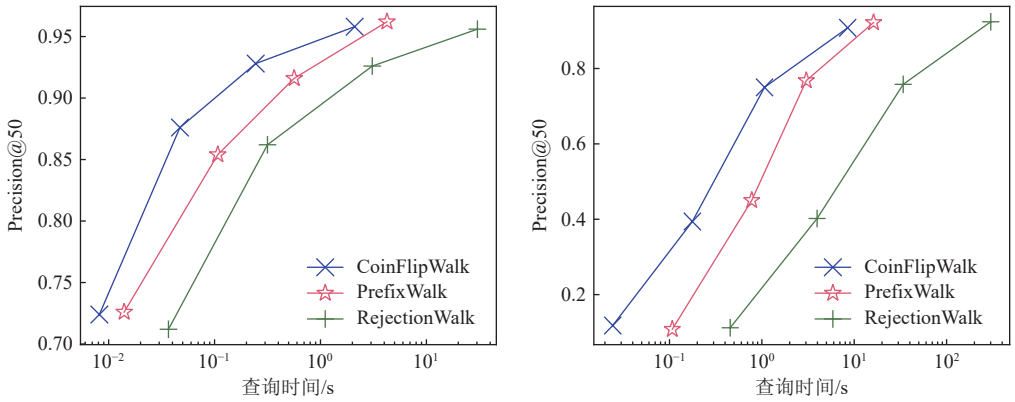
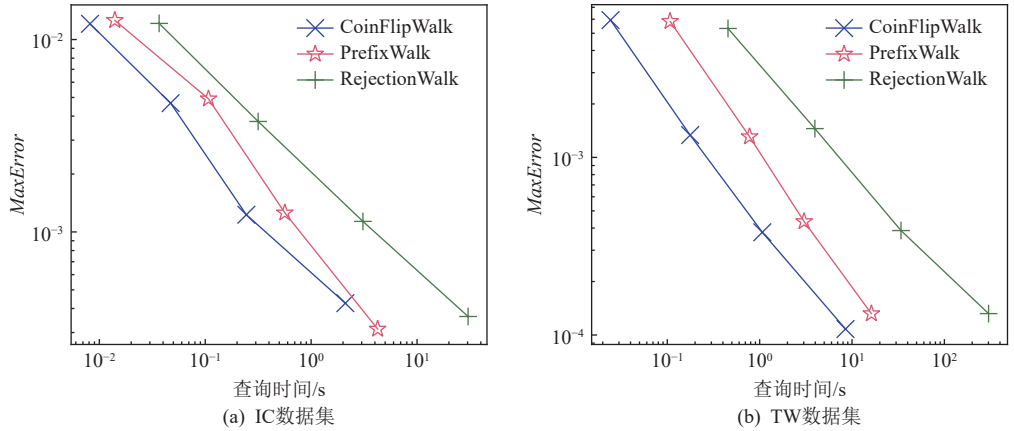


Fig. 5 Precision@50 of different methods varies with query time on synthetic weighted graphs

图 5 在合成有权图上各方法的 Precision@50 随查询时间的变化



(a) IC数据集 (b) TW数据集

Fig. 6 *MaxError* of different methods varies with query time on synthetic weighted graphs

图 6 在合成有权图上各方法的 *MaxError* 随查询时间的变化

5.4 更新时间分析

本节对各方法的随机游走概率更新时间进行了对比. 其中, 图 8 展示了各方法在真实有权图上的随机游走概率更新时间, 图 9 展示了各方法在合成有权图上的随机游走概率更新时间. 由图 8 和图 9 可以看出, 本文所提 CoinFlipWalk 算法在各数据集上所需的更新时间均小于 PrefixWalk 和 RejectionWalk 算法, 这与本文理论分析所得结论一致, 证明了 CoinFlipWalk 算法的高效性. 此外, 由图 8 和图 9 还可看出, PrefixWalk 的更新时间开销显著大于 CoinFlipWalk 和 RejectionWalk, 这一点亦与本文分析吻合, 即前缀和方法每次边插入/删除操作都需要 $O(\log n)$ 的更新时间复杂度. 相比之下, CoinFlipWalk 和 RejectionWalk 的更新时间复杂度仅为 $O(1)$ 和 $O(\log\log n)$.

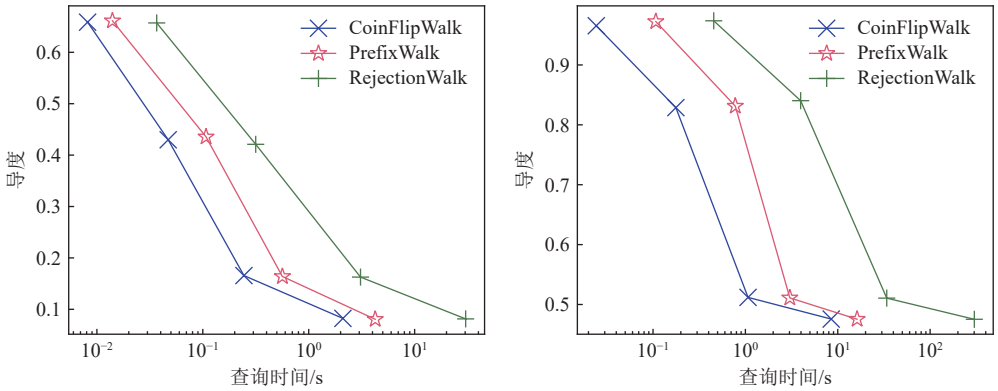


Fig. 7 Conductance of different methods varies with query time on synthetic weighted graphs

图 7 在合成有权图上各方法的导度随查询时间的变化

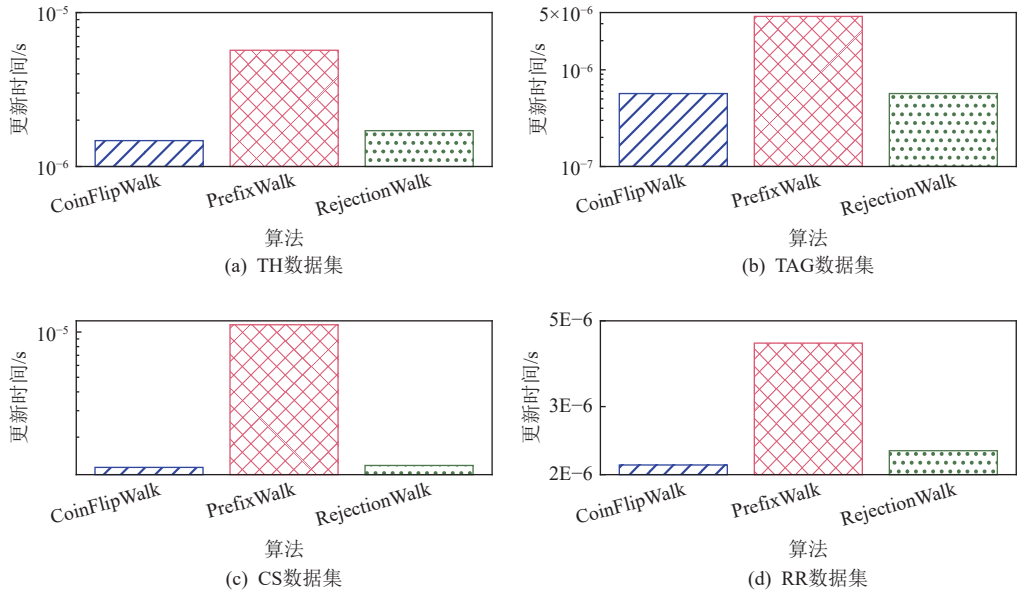


Fig. 8 Update time of different methods on real-world weighted graphs

图 8 在真实有权图上各方法的更新时间

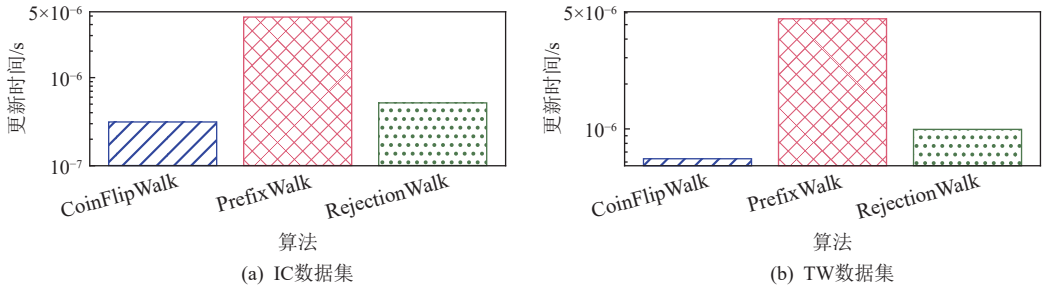


Fig. 9 Update time of different methods on synthetic weighted graphs

图 9 在合成有权图上各方法的更新时间

6 结 论

本文研究了动态有权图上随机游走概率的近似计算问题，提出了一种基于硬币翻转采样的随机游走概率算法. 相比于传统的基于权重采样的随机游走概率计算方法，本文所提方法在随机游走概率计算复杂度和采样结构的更新复杂度两方面都具有优势. 未来或可基于 2 个问题展开研究：1)硬币翻转采样技术是否可以进一步优化？2)单点随机游走计算

也是相关领域的核心问题之一,所提方法是否可以应用于单点随机游走概率上的高效计算?

作者贡献声明:王涵之负责算法设计;易璐负责实验验证;魏哲巍和甘骏豪负责理论优化;袁野、文继荣和杜小勇提出论文修改意见. 王涵之和易璐是共同第一作者.

参 考 文 献

[1] Banerjee S, Lofgren P. Fast bidirectional probability estimation in Markov models[J]. arXiv:1507.05998v1

[2] Fogaras D and Rácz B. Scaling link-based similarity search[C]//Proc of the 14th Int Conf on World Wide Web (WWW '05). New York: ACM, 2005: 641–650

[3] Fogaras D, Rácz B, Csalogány K, et al. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments[J]. Internet Mathematics, 2005, 2(3): 333−358

[4] Kallaugher J, Kapralov M, Price E. Simulating random walks in random streams[C]//Proc of the 2022 Annual ACM-SIAM Symp on Discrete Algorithms (SODA). New York: Society for Industrial and Applied Mathematics, 2022: 3091−3126

[5] Li Ronghua, Yu J X, Lu Qin, et al. On random walk based graph sampling[C]/Proc of IEEE 31st Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2015: 927−938

[6] Nazi A, Zhou Z, Thirumuruganathan S, et al. Walk, Not Wait: Faster sampling over online social networks[J]. arXiv:1410.7833v2

[7] Wang S, Yang R, Xiao X, et al. I: Simple and effective approximate single-source personalized pagerank[C]//Proc of the 23rd ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2017: 505−514

[8] Chen M, Wei Z, Ding B, et al. Scalable graph neural networks via bidirectional propagation[J]. Advances in Neural Information Processing Systems, 2020, 33: 14556−14566

[9] Perozzi B, Al-Rfou R, and Skiena S. DeepWalk: Online learning of social representations[C]//Proc of the 20th ACM SIGKDD Int Conf on Knowledge discovery and data mining (KDD '14). New York: ACM, 2014: 701–710

[10] Wu F, Souza A, Zhang T, et al. Simplifying graph convolutional networks[C]//Proc of Int Conf on Machine Learning, Long Beach, California: PMLR, 2019: 6861–6871

[11] Wang H, He M, Wei Z, et al. Approximate graph propagation[C]// Proc of the 27th ACM SIGKDD Conf on Knowledge Discovery & Data Mining. New York: ACM, 2021: 1686−1696

[12] Yang R, Xiao X, Wei Z, et al. 2019. Efficient estimation of heat kernel PageRank for local clustering[C]//Proc of the 2019 Int Conf on Management of Data (SIGMOD'19). New York: ACM, 2019: 1339–1356

[13] Guo Jiawen, Bai Qijie, Lin Zhutian, et al. Dynamic heterogeneous network embedding based on non-decreasing temporal random walk[J]. Journal of Computer Research and Development, 2021, 58(8): 1624−1641
(郭佳雯, 白淇介, 林铸天, 等. 基于非递减时序随机游走的动态异质网络嵌入[J]. 计算机研究与发展, 2021, 58(8): 1624−1641)

[14] Zhang Xiaochi, Yu Hua, Gong Xiujun. A random walk based iterative weighted algorithm for sub-graph query[J]. Journal of Computer Research and Development, 2015, 52(12): 2824−2833
(张小驰, 于华, 宫秀军. 一种基于随机游走的迭代加权子图查询算法[J]. 计算机研究与发展, 2015, 52(12): 2824−2833)

[15] Wu X, Pang H, Fan Y, et al. ProbWalk: A random walk approach in weighted graph embedding[J]. Procedia Computer Science, 2021, 183: 683−689

[16] Jung J, Shin K, Sael L, et al. Random walk with restart on large graphs using block elimination[J]. ACM Transactions on Database Systems, 2016, 41(2): 1−43

[17] Wong C K, Easton M C. An efficient method for weighted sampling without replacement[J]. SIAM Journal on Computing, 1980, 9(1): 111−113

[18] Walker A J. New fast method for generating discrete random numbers with arbitrary frequency distributions[J]. Electronics Letters, 1974, 10(8): 127−128

[19] Walker A J. An efficient method for generating discrete random variables with general distributions[J]. ACM Transactions on Mathematical Software, 1977, 3(3): 253−256

[20] Casella G, Robert C P, Wells M T. Generalized accept-reject sampling schemes[J]. Lecture notes-monograph series, 2004: 342−347

[21] Guo Q, Wang S, Wei Z, et al. Influence maximization revisited: Efficient sampling with bound tightened[J]. ACM Transactions on Database Systems, 2022, 47(3): 1−45

[22] Yi L, Wang H, Wei Z. Optimal dynamic subset sampling: Theory and applications[C]//Proc of the 29th ACM SIGKDD Conf on Knowledge Discovery & Data Mining. New York: ACM, 2023: 3116–3127

[23] Bhattacharya S, Kiss P, Sidford A, et al. Near-optimal dynamic rounding of fractional matchings in bipartite graphs[J]. arXiv preprint, arXiv: 2306.11828, 2023

[24] Mitzenmacher M, Upfal E. Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis[M]. Cambridge, UK: Cambridge University Press, 2017

[25] Tsai M T, Wang D W, Liau C J, et al. Heterogeneous subset sampling[C]//Proc of Computing and Combinatorics: 16th Annual Int Conf. Berlin: Springer, 2010: 500−509

[26] Bringmann K, Panagiotou K. Efficient sampling methods for discrete distributions[C]//Proc of the 39th Int Colloquium (ICALP 2012). Berlin: Springer, 2012: 133−144

[27] Benson A R, Abebe R, Schaub M T, et al. Simplicial closure and higher-order link prediction[J]. Proceedings of the National Academy of Sciences, 2018, 115(48): E11221−E11230

[28] Hessel J, Tan C, Lee L. Science, askscience, and badscience: On the coexistence of highly related communities[C]//Proc of the Int AAAI Conf on web and social media. Palo Alto, CA: AAAI, 2016: 171–180

[29] Liu P, Benson A R, Charikar M. Sampling methods for counting temporal motifs[C]//Proc of the 12th ACM Int Conf on Web Search and Data Mining. New York: ACM, 2019: 294−302

[30] Kumar R, Liu P, Charikar M, et al. Retrieving top weighted triangles in graphs[C]//Proc of the 13th Int Conf on Web Search and Data

Mining. New York: ACM, 2020: 295−303

[31] Boldi P and Vigna S. 2004. The webgraph framework I: Compression techniques[C]//Proc of the 13th Int Conf on World Wide Web (WWW'04). New York: ACM, 595–602

[32] Boldi P, Rosa M, Santini M, et al. 2011. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks[C]//Proc of the 20th Int Conf on World wide web (WWW' 11). New York: ACM, 587–596

[33] Kunegis J. 2013. KONECT: The Koblenz network collection[C]// Proc of the 22nd Int Conf on World Wide Web (WWW'13 Companion). New York: ACM, 1343–1350

[34] Kwak H, Lee C, Park H, et al. 2010. What is Twitter, a social network or a news media? [C]//Proc of the 19th Int Conf on World wide web (WWW '10). New York: ACM, 591–600

[35] Spielman D A, Teng S H. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems[C]//Proc of the 36th Annual ACM Symp on Theory of Computing. New York: ACM, 2004: 81−90

[36] Yin H, Benson A R, Leskovec J, et al. Local higher-order graph clustering[C]//Proc of the 23rd ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2017: 555−564

Wang Hanzhi, born in 1997. PhD candidate. Her main research interest includes graph analysis and learning on large graphs.

王涵之, 1997 年生. 博士研究生. 主要研究方向为大规模图分析与学习.

Yi Lu, born in 2000. PhD student. Her main research interests include graph analysis and learning, dynamic graph neural network,data removal for graph neural network .

易　璐, 2000 年生. 博士研究生. 主要研究方向为图分析与学习、动态图神经网络、图神经网络数据移除.

Wei Zhewei, born in 1986. PhD, professor. His main research interests include big data and machine learning algorithms, graph analysis and learning, and streaming algorithms.

魏哲巍, 1986 年生. 博士, 教授. 主要研究方向为大数据与机器学习算法、图分析与图学习、数据流算法.

Gan Junhao, born in 1988. PhD, senior lecturer. His main research interests include practical algorithms with non-trivial theoretical guarantees for solving problems on massive data.

甘骏豪, 1988 年生. 博士, 高级讲师. 主要研究方向为带有理论保障的大数据算法.

Yuan Ye, born in 1981. PhD, professor. His main research interests include big data management and analysis, artificial intelligence, and distributed computing.

袁　野, 1981 年生. 博士, 教授. 主要研究方向为大数据管理与分析、人工智能、分布式计算.

Wen Jirong, born in 1972. PhD, professor. His main research interests include information retrieval, data mining, and machine learning.

文继荣, 1972 年生. 博士, 教授. 主要研究方向为信息检索、数据挖掘、机器学习.

Du Xiaoyong, born in 1963. PhD, professor. His main research interests include database system, and big data management and analysis.

杜小勇, 1963 年生. 博士, 教授. 主要研究方向为数据库系统、大数据管理与分析.