大规模图上的 SimRank 计算研究综述

张良富 李翠平 陈 红

(中国人民大学信息学院 北京 100872)

[2)](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)

摘 要 SimRank 是一种衡量有向图中任意两节点间结构相似度的模型,其主要思想为,若图中两个节点被相似节点引用,则这两个节点相似. SimRank 计算的相似度被广泛应用到网络图聚类、近似查询和协同过滤等领域. SimRank 计算模型是一个递归模型,其计算时间、空间复杂度非常高,很难应用于大规模图计算. 过去十几年,研究者们针对大规模图提出了许多高效或近似计算的 SimRank 计算算法. 本文首先介绍 SimRank 模型的描述,以及常见的 SimRank 计算问题定义,然后按照计算方式将这些算法分为迭代法、非迭代法与随机游走法三类;将非迭代法分为基于矩阵运算求解、基于节点对图求解以及基于线性表示求解,将随机游走法分为基于不同索引结构求解、基于不同抽样方式求解以及其他随机游走算法;介绍了这些算法的基本概念、计算原理以及算法特点;分析了随机游走法与迭代法、非迭代法之间的关系;对各种算法的时间复杂度、空间复杂度、计算精确度以及可扩展性进行了论述;在此基础总结了这些 SimRank 算法所对应的计算场景,主要包括单点对/单源(Single Pair/Single Source)查询问题、全体/部分节点对(All Pair/Partial Pair)计算问题以及查询问题. 最后对不同算法实验中图的规模进行了总结,并对大规模图上的 SimRank 计算方法进行了总结和展望.

关键词 结构相似度;SimRank 计算;随机游走;算法分析;复杂度分析

中图法分类号 TP391 **DOI 号** 10.11897/SP.J.1016.2019.02665

Research Progress of SimRank Computation on Big Graph: A Survey

ZHANG Liang-Fu LI Cui-Ping CHEN Hong

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

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

Abstract SimRank is a model for measuring the similarity of two vertices in a directed graph. The main idea is that, if two vertices in the same graph are referenced by similar vertices, the two vertices are similar. SimRank scores are widely used in graph clustering, approximate query and collaborative filtering. As SimRank model is a recursive model, its computational time and space complexity of SimRank is very high. So it is difficult to apply the model to large-scale graphs. Over the past decades, researchers have proposed many efficient or approximate computational SimRank calculation algorithms for large-scale graphs. In this paper, we first introduce the definition of SimRank, and the definitions of common SimRank calculation problems. Then we introduce these algorithms and divide these algorithms into three categories: iterative algorithms, non-iterative algorithms and random walk algorithms. Furthermore, we divide the non-iterative algorithms into matrix operation based algorithms, on node-pair graph based algorithms and linear representation based algorithms, and divide the random walk algorithms into index based algorithms,

收稿日期:2018-07-16;在线出版日期:2019-03-15. 本课题得到国家重点研发计划(2018YFB1004401)、国家自然科学基金(61772537,61772536,61702522,61532021)资助.**张良富**,博士研究生,主要研究方向为数据挖掘、机器学习. E-mail: Liangfu_Zhang@ruc.edu.cn.
李翠平(通信作者),博士,教授,主要研究领域为数据挖掘、推荐系统、信息网络分析、流数据管理等. E-mail: licuiping@ruc.edu.cn.
陈 红,博士,教授,主要研究领域为大数据管理与隐私保护、基于新硬件的数据管理与数据分析、数据仓库与数据挖掘等.

sampling methods based algorithms and other random walk algorithms. Meanwhile we introduce basic concepts, calculation principles and algorithm property of these algorithms. The analysis of the relationship between the random walk algorithm and the other two is also conducted. We summarize the time complexity, space complexity and scalability of these algorithms. And we summarize the scenarios where these algorithms are applied, which mainly are Single Pair/Single Source, All Pair/Partial Pair and SimRank Join Query problems. At last, the scale of graphs in experiments of different algorithms is summarized, and the calculation methods of SimRank on large-scale graphs are summarized and forecasted.

Keywords structural similarity; SimRank calculation; random walk; algorithm analysis; complexity analysis

1 引 言

大规模图上的节点相似度计算是一个非常基础的研究问题,广泛应用于很多领域,如推荐系统[1]、协同过滤[2]、链接预测[3]、网络垃圾邮件检测[4]、网络图聚类[5]和自然语言处理[6]等. 节点间的相似度可以通过图的拓扑结构计算得到. 自 1998 年谷歌网页排名算法 PageRank 提出以来,相继有 SimRank[7]、SimFusion[8]、Penetrating-Rank[9]等基于图拓扑结构的相似度计算模型被提出. 其中 SimRank 被认为是一种比较流行的基于有向图拓扑结构的计算图节点相似度的模型. 它的主要思想为,如果两个对象(节点)被相似的对象引用(即有向图中不同节点的入边邻节点相似或相同),那么这两个节点也相似. 该模型由 Jeh 和 Widom 于 2002 年首次提出[7].

SimRank 是一种递归模型,其计算时间、空间代价很大. 尤其在大规模图上进行 SimRank 计算时,计算代价太高. SimRank 计算的研究主要从以下两个方面展开:优化SimRank计算过程,降低时空复杂度;针对大规模图,进行SimRank值的近似快速计算[10].

SimRank 计算方法主要有三种类型:(1)迭代法. 通过设定节点相似度初始值,根据文献[7]中 SimRank 的定义,进行迭代计算. 虽然这种算法的计算结果精度较高,但时空复杂度也较高,其中空间复杂度为 $O(n^2)$,因此难以应用于大规模图;(2)非迭代法. 将 SimRank 计算中的非线性计算近似改为线性计算,从而将 SimRank 计算变为求解线性系统问题[11,19],进行近似计算. 这种方法一般适合中等规模图,计算结果稳定性较好,但由于为近似算法,因此计算精确度较差;(3)随机游走法. 根据 SimRank 模型的随机游走定义方式进行计算,通过使用不同随机游走方式,构建不同随机游走索引,减少存储空间,提高查询效率,从而降低时间空间复杂度;且由于随机游走之间的独立性,以及每个节点产生的随机游走受原图规模影响较小,因此随机游走算法一般具有较好的可扩展性,但随机游走算法计算精度和结果的稳定性受图结构影响较大. 其中文献[10]证明非迭代法求得 SimRank 值为近似值,并通过随机游走解释证明计算出误差范围.

不同的计算方法一般支持不同的查询. 按照被查询节点的数量,SimRank 计算支持的查询主要分为两类:(1)查询全体或者部分节点对(All Pair/Partial Pair)之间的相似度值. 支持这类查询的 SimRank 计算,常采用迭代法与非迭代法,通过矩阵迭代运算或求解线性系统,来计算全体节点或部分节点对之间的 SimRank 值;(2)查询单点对之间或者单源节点(Single Pair/Single Source)到所有其它节点的相似度值. 支持这类查询的 SimRank 计算,常用随机游走算法,通过建立反向随机游走索引,查询随机游走首次相遇概率及步长,计算相似度,其中单源节点查询中较常见的有 Top-k 查询.

本文首先对 SimRank 的模型定义进行描述,并定义常见 SimRank 查询问题;分析非迭代算法与迭代算法 SimRank 值计算相同条件;分析不同方法下 SimRank 算法特点与计算复杂度;最后分析 SimRank 计算研究进展.

2 模型描述与问题定义

本部分将给出描述 SimRank 模型的数学定义及常见 SimRank 计算问题. 首先表 1 中为常用符号及含义.

表 1　常用符号及相关含义

符号	描述
$G(V,E)$	边集合为 E，点集合为 V 的图
$s(a,b)$，$\boldsymbol{S}$	节点相似度与相似度矩阵
$\boldsymbol{S}_k$	经过 k 次迭代计算相似矩阵
$\tilde{\boldsymbol{S}}$ 或 $\boldsymbol{S}'$	相似矩阵近似计算结果
$I(a)$	节点 a 的入边邻节点集合
c	SimRank 计算中阻尼(或衰减)系数
$\boldsymbol{P}$	图的邻接矩阵
$\boldsymbol{Q}$	$\boldsymbol{P}$ 列归一，$\boldsymbol{Q}(i,j)$表示从节点 i 一步游走到 j 的概率
m，n	图中边的数量 m 与图中节点数量 n
T，t	随机游走步数
k	迭代计算次数
d	图的平均入度
$d_{\max}$	图的最大入度
r	矩阵的秩
$\boldsymbol{W}_v$	起点为 v 的随机游走，即 $W_v[0]=v$
P_t	经 t 步随机游走第一次相遇的概率
$\mathbb{E}[x]$	变量 x 的期望
ε	SimRank 计算误差
δ	蒙特卡洛方法计算误差大于 ε 概率小于 δ

2.1　SimRank 模型定义

SimRank 是 Jeh 与 Widom 于 2002 年提出的通过图 $G(V,E)$的拓扑结构衡量图中任意两节点相似度的模型[7]. SimRank 计算满足以下两条规则：(1) 如果两个不同对象被相似对象引用，则这两个对象相似(递归定义)；(2) 每个对象与其自身相似度最高(基本情况).

定义 1.　SimRank 定义的数学表达如下[7]：

$$s(a,b)=\begin{cases}1, & a=b\\ \dfrac{c}{|I(a)|\cdot|I(b)|}\displaystyle\sum_{x\in I(a)}\sum_{y\in I(b)}s(x,y), & a\neq b\\ 0, & I(a)=\varnothing \text{ 或 } I(b)=\varnothing\end{cases}\tag{1}$$

其中，c 为取值 0 到 1 之间的阻尼系数，一般取值范围 0.6～0.8[7,12,37]，$|I(a)|$表示节点 a 的入边邻节点集合中元素数量.

若图 G 邻接矩阵为 $\boldsymbol{P}$，矩阵 $\boldsymbol{P}$ 的列归一化矩阵为 $\boldsymbol{Q}$，则相似矩阵 $\boldsymbol{S}$ 根据表达式(1)可表示为

$$\boldsymbol{S}=(c\cdot\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{S}\boldsymbol{Q})\vee\boldsymbol{I}\tag{2}$$

其中，$\boldsymbol{Q}^{\mathrm{T}}$为矩阵 $\boldsymbol{Q}$ 的转置，$\boldsymbol{I}$ 表示单位矩阵，操作符“∨”表示取左右矩阵中对应元素最大值，即将计算结果对角元素均取值为 1.

定义 2.　基于随机游走模型的 SimRank 定义[7]，两节点 a、b 的相似度等于以阻尼(衰减)系数 c 为基，以从这两节点出发的反向随机游走首次相遇所经过的步数为指数的函数的期望. 具体数学描述如下，若反向随机游走 $W_a=\{a_0,a_1,\cdots,a_t\}$，$W_b=\{b_0,b_1,\cdots,b_t\}$分别为从节点 a，b 出发，步长为 t 的反向随机游走，即 $a_0=a$、$b_0=b$，$a_i\in I(a_{i-1})$、$b_i\in I(b_{i-1})$，

$$s(a,b)=\mathbb{E}[c^t]=\sum_{t=0}^{\infty}c^t\sum_{x\in V}P_t(a,b,x)\tag{3}$$

其中，t 为随机游走步长，$P_t(a,b,x)$表示随机游走 W_a、W_b经过 t 步首次在节点 x 相遇的概率，即 $a_t=b_t=x\in V$ 且 $a_i\neq b_i(i<t)$的概率，$\mathbb{E}[c^t]$表示 c^t期望. 若阻尼系数相同，定义 1 与定义 2 得到相似值相同，详细证明见文献[7].

例 1.　计算图 1 中节点 SimRank 相似度，其中 $c=0.6$.

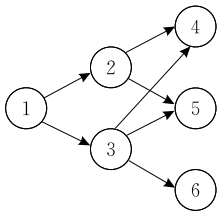

图 1　示例图

根据定义 1，每个节点跟自己相似度为 1，由于节点 1 没有入边，因此节点 1 与任何节点相似度为 0，$s(2,3)=\dfrac{c}{1\cdot1}(s(1,1))=0.6$，$s(4,5)=\dfrac{c}{2\cdot2}(s(2,2)+s(2,3)+s(3,2)+s(3,3))=0.48$，$s(5,6)=\dfrac{c}{2\cdot1}(s(2,3)+s(3,3))=0.48$，$s(4,6)=\dfrac{c}{2\cdot1}(s(2,3)+s(3,3))=0.48$，其他节点间相似度为 0.

根据定义 2，每个节点与自己的相似度根据随机游走步长为 0 计算，由于步长为 0 时随机游走相遇在出发点概率为 1，所以 $s(i,i)=1\cdot c^0=1$；从节点 2、3 出发分别只有一条反向随机游走 2←1 与 3←1，首次相遇在节点 1，相遇步长为 1，所以 $s(2,3)=\dfrac{1}{1}\cdot\dfrac{1}{1}c^1=0.6$；从节点 4 出发反向随机游走 4←2←1，4←3←1，从节点 6 出发反向随机游走 6←3←1，从节点 5 出发反向随机游走有 5←2←1、5←3←1，因此 $s(4,5)=\dfrac{1}{2}\dfrac{1}{2}c^2+\dfrac{1}{2}\dfrac{1}{2}c^2+\dfrac{1}{2}\dfrac{1}{2}c^1+\dfrac{1}{2}\dfrac{1}{2}c^1=0.48$，同理 $s(5,6)=0.48$，$s(4,6)=0.48$.

2.2　SimRank 计算问题定义

按照查询节点数量的不同，SimRank 计算分为全体节点对计算、部分节点对计算、单节点对计算与单源节点相似度查询，其中单源节点相似度计算常用于 Top-k 查询.

1. 全体节点对 SimRank 计算.

输入：有向图 $G(V,E)$，阻尼系数 c，计算误差 ε.

输出：全体节点间相似矩阵 $\boldsymbol{S}'$且 $\max(|\boldsymbol{S}-\boldsymbol{S}'|)\leq\varepsilon$.

2. 部分节点对 SimRank 计算.

输入：有向图 $G(V,E)$，阻尼系数 c；节点集合 $V_1 \subset V$，$V_2 \subset V$.

输出：V_1、V_2 间节点相似集 $\{s(x,y)\}_{\forall x \in V_1, \forall y \in V_2}$.

3. 单点对相似度计算.

输入：有向图 $G(V,E)$，阻尼系数 c，节点 a，b，计算误差 ε.

输出：节点 a，b 间相似度 $s'(a,b)$ 且 $|s'-s| \leq \varepsilon$.

4. 单源节点相似度计算.

输入：有向图 $G(V,E)$，阻尼系数 c，节点 a，计算误差 ε.

输出：与节点 a 相似度非零节点集合 $x \in V$，及相似度 $\{s'(a,x)\}_{\forall x \in V}$，且 $s'-s \leq \varepsilon$.

5. 单源节点 Top-k 相似度计算查询.

输入：有向图 $G(V,E)$，阻尼系数 c，单源节点 a 及节点数量 k.

输出：与 a 相似度最高的节点序列 $\{v_1, v_2, \cdots, v_k\}_{v_i \in V}$.

6. 全图节点对 Top-k Join 计算查询

输入：有向图 $G(V,E)$，阻尼系数 c，节点对数量 k.

输出：k 组全图相似度最高节点对 $\{(v_i, v_j)_k\}_{i<j}$.

问题 6 常见变形问题有输入为有向图(无向图)与某一阈值，输出为全图中相似度大于此阈值的全体节点对.

3 SimRank 计算方法

3.1 迭代法

迭代法根据 SimRank 定义 1，设定初始值 $\boldsymbol{S}_0 = \boldsymbol{I}$，使用式(4)进行迭代计算，直至矩阵 $\boldsymbol{S}_k$ 收敛.

$$\boldsymbol{S}_k = (c \cdot \boldsymbol{Q}^{\mathrm{T}} \boldsymbol{S}_{k-1} \boldsymbol{Q}) \vee \boldsymbol{I} \tag{4}$$

3.1.1 基于原图 G 求解与优化

Jeh 与 Widom 在文献[7]中首先提出 Naïve-SR 算法：每个节点与自己相似度为 1，设置初始 SimRank 矩阵为 $\boldsymbol{S}_0 = \boldsymbol{I}$；根据式(4)迭代计算，直至收敛，并且文献[7]证明了此算法收敛性. 此算法计算时间复杂度为 $O(kd^2n^2)$，空间复杂度为 $O(n^2)$.

为降低计算复杂度，文献[17]提出 SOR-SR 算法，通过行压缩技术存储稀疏邻接矩阵，降低空间复杂度；对于迭代过程，使用连续过松弛(SOR)算法对 SimRank 矩阵进行更新. 对于稀疏矩阵，由于每次迭代只计算邻接矩阵中数值非零节点，计算时间复杂度为 $O(mn)$；对于非稀疏邻接矩阵使用快速矩阵乘法运算，时间复杂度为 $O(n^r)$，其中 r 为常数，且 $1<r<2$. 因此每次迭代 SOR-SR 算法的时间复杂度为 $O(\min\{mn, n^r\})$，空间复杂度为 $O(m+n)$，m 为图中边的数量. 针对 Naïve-SR 算法收敛速度较慢的问题，文献[18]还提出“平方缓存”技术，减少计算次数，加速计算收敛速度；同理，可以使用“n 次方缓存”技术加速计算.

Naïve-SR 算法中矩阵 $\boldsymbol{S}_k$ 为对称矩阵，且不同节点入边邻节点重叠，计算过程中重复计算较多，文献[12,37]提出 Psum-SR 算法，每次迭代通过缓存 $\mathrm{Partial}^k_{x \in I(a)}(x) = \sum_{y \in I(b)} s_k(x,y)$，从而减少重复计算邻域节点间相似度求和次数，降低时间复杂度. 如例 1 中，$I(4) \cap I(5) = \{2,3\}$，计算 $s(4,6)$ 时首先计算存储 $\mathrm{Partial}_{\{2,3\}}(2) = \mathrm{Partial}_{\{2,3\}}(3) = 1.6$，然后计算 $s(4,6) = \frac{c}{|I(4)| \cdot |I(6)|} \sum_{x \in I(6)} \mathrm{Partial}_{I(4)}(x) = 0.48$，计算 $s(5,6) = \frac{c}{|I(5)| \cdot |I(6)|} \sum_{x \in I(6)} \mathrm{Partial}_{\{2,3\}}(x) = 0.48$.

Psum-SR 算法由原来两层循环变为一层循环，每次迭代时间复杂度为 $O(mn)$，由于存储 $\mathrm{Partial}_{I(a)}(x)$ 值需要额外 $O(n)$ 空间，而中间结果 $\boldsymbol{S}_k$ 占用空间为 $O(n^2)$，因此空间复杂度为 $O(n^2)$. 文献[12]中使用关键节点选取技术只针对具有入边邻接点集合的节点进行计算，并且使用节点空间划分方法将节点空间划分为 $\frac{n}{\log_2 n}$ 个子集计算，从而将计算时间复杂度降低为 $O\left(\min\left\{kmn, k\frac{n^3}{\log_2 n}\right\}\right)$.

文献[13]提出的 OIP-DMST 算法通过改变计算 $\mathrm{Partial}_{I(a)}(x)$ 与 $\mathrm{Partial}_{I(b)}(x)$ 的顺序，进一步减少计算量. 如例 1，通过首先计算 $\mathrm{Partial}_{I(6)}(x)$，再计算 $\mathrm{Partial}_{I(5)}(x) = \mathrm{Partial}_{I(6)}(x) + s(2,x)$ 从而减少计算 $\mathrm{Partial}_{I(5)}(x)$ 的时间复杂度. OIP-DMST 算法首先需要生成节点对图，然后生成最小生成树，从而确定 $\mathrm{Partial}_{I(a)}(x)$ 计算顺序，因此时间复杂度为 $O(kd'n^2)$，其中 $d' \leq d$，与公共入邻边数量有关；另外，文献[13]中 SimRank 计算迭代式近似表示为幂级数形式，并且证明其渐进收敛性相似，从而通过幂级数形式估算确定精度下迭代次数 $k' \ll k$，使得迭代次数大幅度减少，从而进一步提高计算效率.

3.1.2 迭代算法总结

迭代算法首先预置相似度矩阵，根据 SimRank 定义进行迭代计算，如 Naïve 算法；由于图邻接矩阵比较稀疏，因此迭代过程中通过稀疏矩阵压缩技术，降低计算空间复杂度，同时使用紧密矩阵计算技术降低计算复杂度，另外矩阵迭代计算过程使用连续过度松弛算法，减少计算量，如 SOR-SR 算法；根

据 SimRank 计算的定义，在迭代过程中会多次使用部分节点相似度的和，因此在 Psum-SR 算法与 OIP-DMST 算法中，每次迭代将部分常用相似度和存储，多次使用，从而减少计算量，降低迭代过程计算复杂度.

迭代算法能够直接求解全体节点对相似度，计算精度较高，计算结果稳定性较好.但是迭代算法计算空间复杂度为 $O(n^2)$，不同算法计算时间复杂度不同，但是均高于线性时间复杂度，因此少用于大规模图的 SimRank 计算；另外由于迭代算法迭代过程每步迭代计算过程均基于前一步计算的所有节点，因此当图发生变化时，需要全部重新计算，而迭代算法的计算时间复杂度较高，因此迭代算法不适用于动态图的计算；另外由于迭代算法计算过程需要基于全体节点对相似度，因此需要将全部节点间相似度进行存储，故在迭代算法大图 SimRank 计算上的可扩展性较差.本部分涉及的迭代算法及算法复杂度和扩展性总结如表 2.

表 2　此部分涉及算法及复杂度分析(可扩展性顺序"好","较好","一般","较差","差"，其中 $d'\ll d, k'<k$)

算法	问题	时间复杂度	空间复杂度	准确度	可扩展性
Naïve[7]	All-Pair	$O(kd^2n^2)$	$O(n^2)$	c^k	差
SOR-SR[17]	All-Pair	$O(\min(kmn, kn^r))$	$O(m+n)$	c^k	差
Psum-SR[12]	All-Pair	$O\left(k\min\left(mn, \frac{n^3}{\log_2 n}\right)\right)$	$O(n^2)$	c^k	差
OIP-DMST[13]	All-Pair	$O(k'd'n^2)$	$O(n)$	c^k	差

3.2　非迭代法

在迭代法中，由于存在非线性操作符"∨"，SimRank 迭代计算难以使用线性代数优化算法(如矩阵的低秩近似)进行运算，文献[24,33]指出，SimRank 计算递归式可以作如下线性递归式进行表示：

$$\boldsymbol{S}=(c\cdot\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{S}\boldsymbol{Q})+\boldsymbol{D},$$

其对应线性展开式为

$$\boldsymbol{S}=\boldsymbol{D}+c\cdot\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{Q}+c^2\cdot(\boldsymbol{Q}^{\mathrm{T}})^2\boldsymbol{D}\boldsymbol{Q}^2+\cdots$$

其中 $\boldsymbol{D}$ 为对角修正矩阵，$\boldsymbol{D}=\boldsymbol{S}-(c\cdot\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{S}\boldsymbol{Q})$，且 $(1-c)\leqslant d_{ii}\leqslant 1$.如图 1 中，$c=0.6$ 时，$\boldsymbol{D}=\mathrm{diag}(1, 0.4, 0.4, 0.52, 0.52, 0.4)$.在实际计算过程中由于矩阵 $\boldsymbol{D}$ 难以直接求解，文献[11,17,19,26,30]中均使用 $(1-c)\boldsymbol{I}$ 替代矩阵 $\boldsymbol{D}$，虽然计算 SimRank 值与定义 1 的 SimRank 值不相同，但是对 Top-k 顺序影响不大，因此可用于 Top-k 查询计算.文献[11,17,19,26,30]中近似表达式为

$$\boldsymbol{S}=(c\cdot\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{S}\boldsymbol{Q})+(1-c)\boldsymbol{I} \tag{5}$$

对应线性展开式为

$$\boldsymbol{S}=(1-c)\cdot\sum_{k=0}^{\infty}c^k\cdot(\boldsymbol{Q}^{\mathrm{T}})^k\boldsymbol{Q}^k.$$

3.2.1　矩阵运算直接求解

由于式(5)为线性系统，因此可以使用求解线性系统的高效的近似算法进行求解计算.文献[11,19]在 SimRank 近似计算式(5)基础上，使用线性代数手段对线性系统进行求解，提出非迭代算法 NI-SR，并将 NI-SR 算法扩展到 GPU 计算[30]，大大提高了矩阵计算效率.

NI-SR 算法基于矩阵 SVD 分解与 Kronecker 乘积等性质，将 SimRank 计算查询过程分为两个过程，预处理过程与查询过程.其中预处理过程根据下式计算：

$$\mathrm{Vec}(\boldsymbol{S})=(1-c)(\boldsymbol{I}-c(\boldsymbol{Q}\otimes\boldsymbol{Q}))^{-1}\mathrm{Vec}(\boldsymbol{I}).$$

上式首先对矩阵 $\boldsymbol{Q}$ 使用 SVD 分解，然后进行求解，从而降低计算量.预处理的时间复杂度为 $O(k^4n^2)$，其中 k 为奇异值分解后前 k 个奇异值，因此空间复杂度为 $O(k^2n^2)$.由于此过程线下进行，因此对于时间空间复杂度敏感性较低.查询过程根据下式计算：

$$\boldsymbol{S}(i,j)=(1-c)(\boldsymbol{I}(i,j)+c\boldsymbol{V}_l\boldsymbol{V}_r),$$

其中，$\boldsymbol{V}_l=\boldsymbol{K}_u((i-1)n+j,:)\boldsymbol{\Lambda}$，$\boldsymbol{K}_u$ 与 $\boldsymbol{\Lambda}$ 根据文献[11]通过矩阵 $\boldsymbol{Q}$ 使用 SVD 分解与 Kronecker 积计算得到，因此查询阶段的时间复杂度为 $O(k^4)$，空间复杂度为 $O(k^2n^2)$.

对于随时间动态变化的图(节点数不变，边数增加)，将增量邻接矩阵作 SVD 分解，然后基于原图计算结果对新图 SimRank 值进行更新，动态更新的时间复杂度与空间复杂度都大大减少，新增时间复杂度至多增加 $O(n^2)$.

由于 NI-SR 算法分为线下计算与线上计算两部分，主要计算时间复杂度在线下进行，且线下计算对时间空间敏感性相对较低，因此可以将此算法推广到中等规模图(10 K～100 K 节点数量)上进行计算；同时，由于计算过程仅涉及矩阵运算，因此可以使用 GPU 加速技术进行计算，从而大大提高计算效

率与计算精度[30];由于主要计算为线下计算,因此,此算法可扩展性增加,但是由于计算中使用 Kronecker 乘积,使得计算空间复杂度大大提高,内存占用量大大增加,此算法很难扩展到大规模图的 SimRank 计算中.

基于式(5),文献[19]提出一种基于西尔维斯特方程[20]求解的算法 SimMat,SimMat 基于矩阵特征分解求解线性系统. SimMat 算法也分为预处理过程与查询过程. 首先将式(5)导为西尔维斯特方程,如下:

$$\frac{1}{c}\boldsymbol{Q}^{-1}\boldsymbol{S}-\boldsymbol{S}\boldsymbol{Q}^{\mathrm{T}}=\frac{1-c}{c}\boldsymbol{Q}^{-1}.$$

然后将 $\boldsymbol{Q}^{\mathrm{T}}$ 进行特征值分解为对角阵 $\boldsymbol{D}$ 与单位正交阵 $\boldsymbol{P}$ 进行求解:

$$\boldsymbol{S}=(\boldsymbol{P}^{\mathrm{T}})^{-1}\boldsymbol{X}\boldsymbol{P}^{-1},$$

其中,$\boldsymbol{X}$ 可通过下式求解:

$$\frac{1}{c}\boldsymbol{D}^{-1}\boldsymbol{X}-\boldsymbol{X}\boldsymbol{D}=\frac{1-c}{c}\boldsymbol{D}^{-1}\boldsymbol{P}^{\mathrm{T}}\boldsymbol{P}.$$

在查询阶段,对单点查询 $s(a,b)=\boldsymbol{l}_a\cdot\boldsymbol{r}_b$,其中 $\boldsymbol{l}_a$、$\boldsymbol{r}_b$ 根据上式计算[19].

SimMat 算法通过将邻接矩阵特征分解出对角阵,预计算得出中间结果并保存,对于要查询的单源问题(如 Top-k 查询),可以选择中间结果进行查询搜索,由于没有使用 Kronecker 乘积,因此中间计算的空间复杂度小于 NI-SR 算法. 预处理时间复杂度为 $O(rn^2)$,其中 r 为矩阵 $\boldsymbol{Q}$ 的秩. 查询时间复杂度为 $O(rn)$,空间复杂度为 $O(rn^2)$. 由于计算过程需要进行矩阵特征分解,而图的邻接矩阵并不是均能可对角化,因此 SimMat 算法在某些情况下难以得出结果[8].

3.2.2 基于节点对图 G^2 精确求解

由于 $\boldsymbol{D}\neq(1-c)\boldsymbol{I}$,因此前面方法中计算结果均非 SimRank 定义中的 SimRank 值,所以文献[32]与文献[34]分别提出基于 G^2 的 SimRank 线性表示,然后分别基于提出的线性表达式进行求解.

定义 3. 节点对图. 图 $G(V,E)$ 对应的节点对图 $G^2(V^2,E^2)$ 定义如下:

(1) 每个节点 $n_i\in V^2$ 表示图 G 中一个节点对 $n_i=\langle x_i,y_i\rangle$,其中 $x_i,y_i\in V$.

(2) 图 G^2 中边 $(n_{i1},n_{i2})\in E^2$ 存在,当且仅当 $(x_{i1},x_{i2})\in E$,$(y_{i1},y_{i2})\in E$,其中 $n_{i1}=\langle x_{i1},y_{i1}\rangle$,$n_{i2}=\langle x_{i2},y_{i2}\rangle$.

文献[32]基于 G^2 图将定义 1 的非线性表达式表示为封闭的线性系. 令 $V_{\boldsymbol{S}}^2=\{n_i\mid n_i=\langle x_i,x_i\rangle,x_i\in G,n_i\in G^2\}$,$V_{\boldsymbol{D}}^2=\{n_i\mid n_i=\langle x_i,x_j\rangle,x_i,x_j\in G,x_i\neq x_j,n_i\in G^2\}$,则 $V_{\boldsymbol{D}}^2$、$V_{\boldsymbol{S}}^2\subset V^2$,且 $V_{\boldsymbol{D}}^2\cap V_{\boldsymbol{S}}^2=\varnothing$,$V_{\boldsymbol{D}}^2\cup V_{\boldsymbol{S}}^2=V^2$;令 $I(n_i)$ 表示节点 n_i 在图 G^2 中的入边邻节点节点,$I_{\boldsymbol{S}}(n_i)=I(n_i)\cap V_{\boldsymbol{S}}^2$,$I_{\boldsymbol{D}}(n_i)=I(n_i)\cap V_{\boldsymbol{D}}^2$. 图 G 中节点相似度 $s(x_i,y_i)$ 为 G^2 中 $s(n_i)$,$n_i=\langle x_i,y_i\rangle$,则定义 1 中非线性表达式可表示如下[32]:

$$s(n_i)=\begin{cases}\dfrac{c}{|I(n_i)|}\Big(\displaystyle\sum_{n_j\in I_{\boldsymbol{D}}(n_i)}s(n_j)+|I_s(n_i)|\Big), & n_i\in V_{\boldsymbol{D}}^2\\ 1, & n_i\in V_{\boldsymbol{S}}^2\end{cases},$$

整理如下式

$$|I(n_i)|s(n_i)-c\sum_{n_j\in I_{\boldsymbol{D}}(n_i)}s(n_j)=c|I_s(n_i)|.$$

令 $\boldsymbol{A}_{\boldsymbol{S}}(i,j)=\begin{cases}1, & j\in I_{\boldsymbol{S}}(i)\\ 0, & \text{其他}\end{cases}$,$\boldsymbol{A}_{\boldsymbol{D}}(i,j)=\begin{cases}1, & j\in I_{\boldsymbol{D}}(i)\\ 0, & \text{其他}\end{cases}$,$\boldsymbol{I}_{\boldsymbol{D}}=\mathrm{Diag}(\mathrm{Vec}(J-I))$,$\boldsymbol{I}_{\boldsymbol{S}}=\mathrm{Diag}(\mathrm{Vec}(I))$,$d_{n_i}=|I(n_i)|$,$v_{n_i}=|I_s(n_i)|$,文献[32]将上式统一整理得下式:

$$(\mathrm{Diag}(\vec{\boldsymbol{d}})-c\boldsymbol{I}_{\boldsymbol{D}}\boldsymbol{A}_{\boldsymbol{D}})\mathrm{Vec}(\boldsymbol{S})=c\boldsymbol{I}_{\boldsymbol{D}}\vec{\boldsymbol{v}}+\boldsymbol{I}_{\boldsymbol{S}}\vec{\boldsymbol{d}}.$$

文献[32]通过使用节点对图,将 SimRank 计算中非线性操作符“∨”消除,将 SimRank 计算问题构造为线性系统求解问题,同时保留计算精度,然后通过共轭梯度法求解这个线性系统,由于 $(\mathrm{Diag}(\vec{\boldsymbol{d}})-c\boldsymbol{I}_{\boldsymbol{D}}\boldsymbol{A}_{\boldsymbol{D}})$ 为严格主对角占优,因此可以使用预置共轭梯度法求解无向图,对于有向图,可以使用双向共轭梯度法求解. 整个计算过程分为预计算过程与计算过程. 首先通过预计算过程计算 $\vec{\boldsymbol{v}}$ 与 $\vec{\boldsymbol{d}}$,与计算过程的时间复杂度为 $O(mn)$,空间复杂度为 $O(n^2)$;计算过程通过预置系统使用共轭梯度法,空间复杂度为 $O(n^2)$,时间复杂度为 $O(kmn)$,其中 $k=\lceil\log_c\varepsilon-1\rceil/\log_c\dfrac{\sqrt{\sigma}-1}{\sqrt{\sigma}+1}$,$\sigma$ 为预置系统矩阵谱半径.

除文献[32]中将 SimRank 计算表达式推导为封闭的线性表达式的方法以外,文献[34]推导为更简洁形式如下:

$$(\boldsymbol{I}-c\boldsymbol{F})\mathrm{Vec}(\boldsymbol{S})=\mathrm{Vec}(\boldsymbol{I}),$$

其中 $\boldsymbol{F}=(\boldsymbol{I}-\mathrm{Diag}(\mathrm{Vec}(\boldsymbol{I})))(\boldsymbol{Q}^{\mathrm{T}}\otimes\boldsymbol{Q}^{\mathrm{T}})$.

然后文献[34]基于前向局部推进算法(FLP-SR),求解上述线性表达式. 此算法的时间复杂度为 $O\Big(\dfrac{d^2}{c(1-c)^2\varepsilon}\Big)$,空间复杂度为 $O(m+n+|dense(\boldsymbol{R})|)$,

$\boldsymbol{R}$ 为残差矩阵，由于 $\boldsymbol{R}$ 为稀疏矩阵，$|dense(\boldsymbol{R})|$ 为残差矩阵的压缩表示空间复杂度.

3.2.3　线性表示求解

非迭代算法中，除使用矩阵求逆方式求解，文献[24]提出线性表达式：

$$\boldsymbol{S}=\boldsymbol{D}+c\cdot\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{Q}+c^2\cdot(\boldsymbol{Q}^{\mathrm{T}})^2\boldsymbol{D}\boldsymbol{Q}^2+\cdots$$

此线性表达式可表示为

$$s^{(k)}(u,v)=D_{uv}+c\,\mathbb{E}[\boldsymbol{e}_u^{(1)}]^{\mathrm{T}}\boldsymbol{D}\,\mathbb{E}[\boldsymbol{e}_v^{(1)}]+\cdots+c^{k-1}\mathbb{E}[\boldsymbol{e}_u^{(k-1)}]^{\mathrm{T}}\boldsymbol{D}\,\mathbb{E}[\boldsymbol{e}_v^{(k-1)}],$$

其中，$\mathbb{E}[\boldsymbol{e}_u^{(k)}]$ 表示从 u 点反向随机游走 k 到达其他各个节点的期望，文献[24]通过蒙特卡洛方法求解上述线性表达式中 $\mathbb{E}[\boldsymbol{e}_u^{(k)}]$. 由线性表达式可以得出，$\boldsymbol{D}$ 的取值对于单源节点 Top-k 查询影响不大，因此文献[24]使用此方法进行单源节点 Top-k 查询计算.

文献[33]在文献[24]基础上，通过高斯—谢尔德算法求解 $\boldsymbol{D}$，提高此算法精度，且通过随机阈值方式随机剔除小于某一阈值的中间结果，减少中间计算结果的时间空间复杂度，从而进一步提高此算法计算效率，且将此算法推广 Top-k join 查询问题.

由近似表达式(5)可得到如下公式：

$$\boldsymbol{S}=(1-c)\sum_{k=0}^{\infty}c^k\cdot(\boldsymbol{Q}^{\mathrm{T}})^k(\boldsymbol{Q})^k,$$

对于某一节 j 的相似度单源查询，便可表示为

$$\boldsymbol{S}[:,j]=(1-c)\left(\boldsymbol{e}_j+\sum_{k=1}^{\infty}c^k\cdot(\boldsymbol{Q}^{\mathrm{T}})^k(\boldsymbol{Q})^{k-1}\boldsymbol{Q}[:,j]\right)$$
$$=(1-c)(\boldsymbol{e}_j+c^1(\boldsymbol{Q}^{\mathrm{T}})^1\boldsymbol{Q}[:,j]+\cdots).$$

因此文献[14]针对单源查询提出“种子生成算法”，将线性展开式中矩阵计算 $(\boldsymbol{Q}^{\mathrm{T}})^k(\boldsymbol{Q})^{k-1}\boldsymbol{Q}[:,j]$ 的计算过程设计为“从右向左”计算，则所有计算均为“矩阵×向量”运算，避免了“矩阵×矩阵”运算，从而降低计算时间复杂度. 同时文献[14]通过设计矩阵计算顺序，大幅度降低上式重复计算次数，从而提高计算效率.

如例 1 中查询节点 4 与其他节点相似度，此公式表示随机游走解释如图 2 所示.

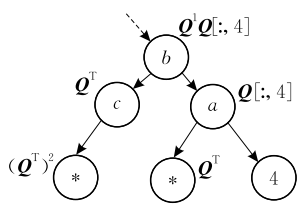

图 2　从节点 4 出发的随机游走

图 2 中，$4\leftarrow a$ 表示从节点 4 反向随机游走到节点 a，其中 a 在例 1 中表示节点 $a=\{2,3\}\subset V$，$\boldsymbol{Q}[:,4]$便为节点 4 到节点 a 的概率向量$[0,0.5,0.5,0,0,0]^{\mathrm{T}}$；由于 $a\subset V$，从节点 a 跳转到其他任意节点概率满足左乘矩阵 $\boldsymbol{Q}^{\mathrm{T}}$，所以 $\boldsymbol{Q}^{\mathrm{T}}\cdot\boldsymbol{Q}[:,4]$ 表示任意节点发出的反向随机游走与从 4 出发的随机游走经过 1 步游走时在节点 a 相遇的概率，所以 $\boldsymbol{S}_1[:,4]=(1-c)(c^0\boldsymbol{e}_4+c^1\boldsymbol{Q}^{\mathrm{T}}\cdot\boldsymbol{Q}[:,4])$，其中 $c^0\boldsymbol{e}_4=[0,0,0,1,0,0]^{\mathrm{T}}$ 表示随机游走经过 0 步跳跃相遇节点的概率；同理，通过“种子生成算法”计算 $(\boldsymbol{Q}^{\mathrm{T}})^2(\boldsymbol{Q}\boldsymbol{Q}[:,4])=\boldsymbol{Q}^{\mathrm{T}}(\boldsymbol{Q}^{\mathrm{T}}(\boldsymbol{Q}\boldsymbol{Q}[:,4]))$，以此类推，便可得到 $\boldsymbol{S}_k[:,4]$.

Par-SR 算法通过下式可进行部分节点相似度计算：

$$\boldsymbol{S}_k(\boldsymbol{A},\boldsymbol{B})=(1-c)\cdot\sum_{k=0}^{\infty}c^k\boldsymbol{Q}^{\mathrm{T}}[\boldsymbol{A},:](\boldsymbol{Q}^{\mathrm{T}})^{k-1}\boldsymbol{Q}^{k-1}\boldsymbol{Q}[:,\boldsymbol{B}].$$

Par-SR 算法进行单源(Top-k)查询时，时间复杂度为 $O(\max_i\{d_i^{2k}\})$，空间复杂度为 $O(m+kn)$.

由于算法 Par-SR 中 $\boldsymbol{Q}^k$ 表示各个节点经过 k 次正向随机游走到达其他各节点的概率，而非首次到达此节点的概率，所以计算得到的 SimRank 值与文献[7]中定义的不同，但对单源查询计算结果影响不大，特别针对 Top-k 查询问题.

根据文献[24]中对于单点对相似度计算 $s(a,b)=D_{ab}+\cdots+c^t\mathbb{E}[\boldsymbol{e}_a^{(t)}]^{\mathrm{T}}\boldsymbol{D}\,\mathbb{E}[\boldsymbol{e}_b^{(t)}]$，对于 $\mathbb{E}[\boldsymbol{e}_b^{(t)}]$的计算，文献[24]通过蒙特卡洛模拟方法分别生成 $r=2(1-c)^2\log(4nt/\delta)/\varepsilon^2$ 组随机游走，其中 t 为随机游走步长，则计算误差大于 ε 概率不高于 δ. 通过蒙特卡洛模拟方法可降低大图中 $(\boldsymbol{Q}^{\mathrm{T}})^k\boldsymbol{D}\boldsymbol{Q}^k$ 的计算时间复杂度，并且在实际实验中随机游走数量与图的规模无关，且能维持较好结果，因此，此算法在大图的可拓展性大大提高.

3.2.4　非迭代法总结

非迭代算法主要通过将 SimRank 计算过程近似表示为线性系统，利用线性代数等手段对线性系统进行求解，如矩阵的 SVD 分解、矩阵的特征分解等，然后根据分解结果进行进一步计算，如 NI-SR 算法、SimMat 算法等. 但是这两种非迭代算法分别有自身缺点，如 NI-SR 算法线下计算过程(预计算)需要进行 Kronecker 乘积，矩阵的空间复杂度变为 $O(r^2n^2)$，难以将此算法推广到节点数为 100 K 以上的图中进行计算；而 SimMat 算法涉及归一化邻接矩阵的特征分解，但并非所有图的邻接矩阵均能进

行对角化处理,因此此算法很多情况下难以得到计算结果[7],所以这种算法的稳定性较差.

PBiCG 算法[32]与 FLP-SR 算法[34]将 SimRank 计算按定义转化到 G^2 图中,消除非线性算子"∨",然后使用不同算法对此线性系统进行求解,从而得到准确度较高的 All-Pair 的 SimRank 值,同时计算过程使用线性迭代方式,而没有使用矩阵分解、矩阵求逆等算法,因此可以将稀疏矩阵通过压缩存储来进行计算,从而大大降低计算时间空间复杂度.

通过代数变换,SimRank 计算迭代过程可进行线性表示,基于此线性表示,结合 SimRank 计算问题,如单源计算查询(Top-k 查询),调整线性表达式中每一项计算顺序,减少每次迭代计算时间复杂度,并对线性表达式进行变形,减少不同项中重复计算次数,从而降低计算复杂度与空间复杂度;或者与随机游走结合,通过蒙特卡洛模拟求解线性表示中需要大量迭代计算才能得到的中间结果,从而避免复杂度较高矩阵迭代计算过程. 本部分涉及的非迭代算法及算法复杂度和扩展性总结如表 3 所示.

表 3 此部分涉及算法及复杂度分析(可扩展性顺序"好","较好","一般","较差","差". 其中 T 为步长,k 为迭代次数,r 为矩阵的秩,r_1 与 r_2 为与 ε 相关的抽样参数,但常被设为常数)

算法	问题	时间复杂度		空间复杂度	准确度	可扩展性
		预处理	查询			
NI-SR[11]	All-Pair	$O(r^4n^2)$	$O(r^4)$	$O(r^2n^2)$	无	一般
SimMat[19]	Top-k	$O(rn^2)$	$O(rn)$	$O(rn^2)$	无	差
Par-Sim[14]	Top-k	$O(km)$	$O(\max_i\{d_i^{2k}\})$	$O(m+kn)$	c^k	较好
MonteCarlo[24]	Top-k	$O(n(r_1+r_2)T)$	$O(r_1T)$	$O(m+nP)$	ε	好
PBiCG[32]	All-Pair	$O(mn)$	$O(kmn)$	$O(n^2)$	c^k	一般
FLP-SR[34]	All-Pair	$O\left(\frac{d^2}{c(1-c)^2\varepsilon}\right)$	$O\left(\frac{d^2}{c(1-c)^2\varepsilon}\right)$	$O(m+n+\lvert dense(\boldsymbol{R})\rvert)$	ε	较好

传统非迭代算法一般应用于中等规模图的 SimRank 计算查询,其中优化后所 Par-SR 算法可在较大规模图进行 Top-k 查询;由于蒙特卡洛模拟方法建立随机游走索引的步长与随机游走数量与图的规模无关,因此可以用于较大规模图的 SimRank 查询.

2018 年文献[34]基于 G^2 提出的 SimRank 计算算法,可以将 SimRank 的全体节点对计算推广到 40 M 节点数量的较大规模图中进行计算,且能保证较好的精度.

3.3 随机游走法

根据 SimRank 定义 2,两节点 SimRank 值可通过式(3)求解. 对有向图中任意两点(a,b),分别为起点生成反向随机游走(W_a,W_b),SimRank 值便是以两组随机游走首次相遇步长以阻尼系数 c 为基底指数函数期望$\mathbb{E}[c^t]$.

基于随机游走的算法一般分为两步进行(除 ProbeSim 算法等),预处理阶段生成反向随机游走索引,查询阶段根据生产反向随机游走索引,通过计算相遇次数与步长,计算节点间相似度.

基于随机游走 SimRank 计算方法,根据生成索引方式的不同、随机游走抽样方式的不同以及查询方式的不同一般分为三类:基于不同索引结构的 SimRank 计算,即通过压缩随机游走索引存储结构,减小预处理存储空间,加快查询计算速度;基于不同抽样方式的 SimRank 计算,即通过改变随机游走抽样方式,提高查询计算精度与速度;基于不同查询方式的 SimRank 计算,即通过构建反向随机游走索引,但是通过正向随机游走进行查询,这种随机游走查询方式在单源 Top-k 查询中,只会计算相似度大于 0 的节点对(反向随机游走相遇)的情况,从而降低计算复杂度.

3.3.1 基于不同索引结构的 SimRank 计算

根据式(3),文献[15]提出基于蒙特卡洛方法的 SimRank 计算. 预处理阶段生成指纹图,查询阶段根据生成指纹图计算节点间相似度.

首先通过随机游走方法,计算得到分别从节点 a、b 反向随机游走集合$\{W_a\}$、$\{W_b\}$,计算两个集合中随机游走,经过相同步数 t 首次相遇的次数 N_t,则式(3)中的 $P_t(a,b,x)=\frac{N_t}{\lvert\{W_a\}\rvert\cdot\lvert\{W_b\}\rvert}$;为减少随机游走存储空间,文献[15]使用指纹图存储随机游走集合,指纹图同时存储反向随机游走节点相同经过步数. 指纹图生成满足下列条件:(1)若指纹图中边(a,b)存在,则节点 a 索引大于节点 b,且 W_a,W_b 在有限步内相遇;(2)若同时有多组节点 b 满足条

件(1)，则选择相遇步长最小的节点；(3) 若同时满足条件(1)，条件(2)，则选择索引值较小节点插入指纹图. 所以指纹图有 n 个节点，至多有 $n-1$ 条边.

图 1 生成指纹图如图 3 所示，图中每组随机游走边的数量被压缩为 3 条，且每组指纹图可以直接读出两节点相遇步数，如 FPG1 中，节点 6 与节点 4 相遇步长为 $\max\{1,1\}=1$，FPG2 中，节点 6 与节点 4 相遇步长为 $\max\{1,2\}=2$，而节点 1 产生的随机游走与任何节点均不相遇.

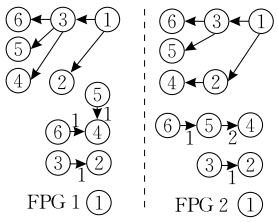

图 3　指纹图

指纹图生成过程比较复杂，计算量较大，文献[21]使用不同的随机游走图结构压缩模型单路图(One-way Graph)，单路图中每个节点只有一条入边，因此每个单路图中每一节点有且只有一条随机游走路径，这种方法将不同节点的随机游走路径耦合到一起，从而减少存储空间，同时这种生成单路图过程避免生成指纹树方法复杂的预计算过程. 在生成的单路图中，TFS 模型可以高效进行节点 Top-k 的 SimRank 查询与单点对 SimRank 查询计算. TSF 模型分为两个步骤，预处理阶段首先将有向图抽样为 r_g 个单路图；查询阶段若为单点对查询 $s(a,b)$，TSF 首先在有向图生成 r_q 条起点为要查询点 a 的反向随机游走，然后在生成单路图中分别生成 r_g 条 b 为起点的随机游走，然后通过式(3)计算求解 $s(a,b)$；对于 Top-k 查询，首先在有向图中生成 r_q 条起点为要查询点 a 的反向随机游走，并记录随机游走过的节点，然后在 r_g 个单路图中以记录的点为起点进行随机游走，这些随机游走经过的点便是与 a 相似的点，然后计算它们与 a 的相似度并选择前 k 个排序. 这种方法有效筛选掉随机游走低概率相遇节点，从而提高计算效率.

如例 1 中途生成单路图如图 4 所示，首先改变图中边的方向生成反向图，然后图中每个节点只选取一条出边，相当于原图中每个节点中仅有一条入边，通过随机抽样生成 r_g 个单路图，如图所示 $r_g=2$. 在查询阶段(Top-k 查询)首先从生成的反向图中随机生成 $r_q=1$ 条以要查询节点 a 为起点的长度为 t 随机游走 $W_4=4\to2\to1$，记录每个经过节点及所需要经过步数{"2":1,"1":2}，然后在单向图中分别以"2"为起点反向游走 1 步即节点"5"，及以"1"为起点反向游走两步得到节点节点"5"、"6"，然后计算以它们为起点随机游走相遇概率为 $s(4,5)=\frac{1}{r_g r_q}(c\cdot 1+c^2\cdot 2)$，$s(4,6)=\frac{1}{r_g r_q}(c^2\cdot 2)$.

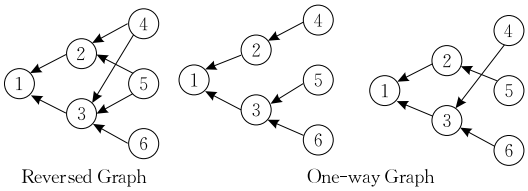

图 4　单路图

由于 TSF 模型 Top-k 查询中所有查询只需查询随机游走能够到达的节点相似度，从而避免许多不必要计算，提高计算效率. 但是这种算法查询考虑所有相遇概率，而非首次相遇概率，因此计算结果与非迭代计算相似，均为近似查询.

基于不同索引结构的 SimRank 计算通过压缩存储索引结构，从而降低索引存储的空间复杂度，以及查询计算时的时间复杂度，但是这种方式每次查询时需要记录相遇步长 t，用于计算 c^t. 而 3.3.2 节中基于不同抽样方式的 SimRank 计算方式能够避免这种情况，只需要计算相同步长下随机游走在索引中相遇次数，便可以进行 SimRank 近似计算.

3.3.2　基于不同抽样方式的 SimRank 计算

一般随机游走方法，首先通过蒙特卡罗方法计算两个节点产生的随机游走相遇的概率，同时计算随机游走相遇步长 t，从而计算相似度 $\mathbb{E}[c^t]$. 文献[16,23,29]根据随机游走模型，提出单路径随机游走方式，即从一个节点 a 反向随机游走到下一入边邻居节点 $b\in I(a)$ 概率为 $\frac{\sqrt{c}}{|I(a)|}$，则此随机路径的概率为 $h^{(t)}(a_0,a_t)=\prod_{k=0}^{t}\frac{\sqrt{c}}{|I(a_k)|}=(\sqrt{c})^t\cdot\prod_{k=0}^{t}\frac{1}{|I(a_k)|}$，根据此定义进行节点相似度 Top-$k$ 查询计算. 文献[16]首次将这种抽样方式定义为 $\sqrt{c}-walk$，并通过此定义推导出一种 SimRank 计算算法. 在查询阶段，这种算法可以直接通过计算两随机游走在相遇次数，从而计算相似度，而不用额外存储相遇步长用来计算 c^t.

定义 4.　对于反向随机游走，到每一节点 $a\in V$，有 $1-\sqrt{c}$ 概率游走停止于 a，而有 $\frac{\sqrt{c}}{|I(a)|}$ 游走到下一个节点 $b\in I(a)$，我们称这种随机游走为 $\sqrt{c}$ 游走.

引理[16]. 令 $h^{(t)}(a,b)$ 表示 $\sqrt{c}$ 游走从 a 进过 t 步到达 b 的概率，d_x 表示从 x 节点出发的 $\sqrt{c}$ 游走经过 0 步后不再相遇的概率，则

$$s(a,b)=\sum_{t=0}^{\infty}\sum_{x\in V}(h^{(t)}(a,x)d_x h^{(t)}(b,x)).$$

证明. 见文献[16].

其中 $d_x=1-\frac{1}{|I(x)|}-\frac{c}{|I(x)|^2}\sum_{\substack{i,j\in I(x)\\ i\neq j}}s(i,j)$.

文献[16]提出使用蒙特卡洛方法，通过两阶段计算节点 SimRank 相似度的算法 SLING. 预处理阶段通过 $\sqrt{c}$ 游走建立索引，查询阶段通过计算两个索引在相同步长下相遇的概率，然后求和计算两节点相似度.

文献[29]将每个节点出发的 $\sqrt{c}$ 游走经过 t 步内到达所有节点的概率修正保存为向量，通过计算两个节点经过相同步长相遇概率和，计算任意节点相似度，$\sqrt{c}$ 游走经过 t 步随机游走到达节点的到达概率使用蒙特卡洛模拟获得，然后修正为 $h^{(t)}(a,x)\sqrt{d_x}$，SimRank 单点对计算只需进行向量点积运算；对于 Top-k 查询，只维护 $2k$ 候选节点进行计算.

文献[22]提出的 READS 算法，预处理阶段首先使用权威游走抽样方式生成 SA-森林，权威游走的首步随机游走停止概率为 0，等概率选择入边邻居节点，从第二步游走开始时使用 $\sqrt{c}$ 游走抽样方式，同时 READS 算法建立图索引时，将图索引结构进行进行合并，从而减小索引空间占用；查询阶段，由于叶子节点存储数组，每棵树的叶子节点在数组中组成循环，便于相似度查询计算.

如图 5，例 1 生成的一组 SA 森林，在同一颗树上的叶子节点相似度非 0，且叶子节点满足同一树上的叶子节点在数组 $Next_i$ 中构成循环，如查询节点“4”的相似点，只需在序列 $Next_i[4-1]=5$，$Next_i[5-1]=6$，$Next_i[6-1]=4$，因此在这组查询中，与节点“4”相似度非 0 节点为{“5”，“6”}.

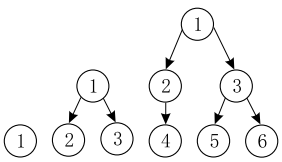

图 5 SA-随机游走与合并的随机游走

基于不同抽样方式的 SimRank 计算通过改变抽样方式，在建立索引的随机游走过程中便保留了步长信息，因此在查询时只需计算相同步长下的相遇概率，便可进行 SimRank 计算查询. 但是在查询过程中，由于多数随机游走并没有相遇，这种方式仍会有许多冗余计算；另外即便随机游走间有相同节点，如果反向随机游走步长不同，SimRank 相似度也为 0. 而 3.3.3 节中的基于不同查询方式的 SimRank 计算，通过只计算随机游走相同步长下相遇节点间相似度，降低查询时间复杂度.

3.3.3 基于不同查询方式的 SimRank 计算

文献[25]提出的 TopSim 算法主要针对单源 Top-k 查询问题，通过维持两个随机游走，使用蒙特卡洛方法计算两随机游走相遇概率，由此计算节点相似度，但是其计算过程未考虑首次相遇情况，因此为近似查询. 算法执行过程为首先根据已知节点 a 生成反向随机游走 t 步到达 a_t，然后，从 a_t 正向随机游走 t 步生成第二个随机游走到达节点 b 生成候选节点集合，通过在候选集合进行 Top-k 计算查询，从而大大减少计算量.

随机游走算法计算时需要首先建立索引，然后进行查询，但是文献[23]基于文献[16]提出关于 SimRank 的单源查询与 Top-k 查询算法 ProbeSim 算法，此算法不需要建立索引，可以通过随机游走，直接进行单源节点相似度“探测”查询.

定义 5[23]. 首次相遇概率. 给定一条反向随机游走路径 $W_a=(a=a_0,a_1,\cdots,a_i)$，$a_i\in I(a_{i-1})$ 及图中节点 $b\in V$ 且 $b\neq a$，则首次相遇概率为

$$\boldsymbol{P}(b,\boldsymbol{W}_a)=\boldsymbol{Pr}_{W_b'}[a_i=b_i,a_{i-1}\neq b_{i-1}\cdots a_0\neq b_0],$$

其中 $W_b'=(b=b_0,b_1,\cdots,b_i)$ 为随机 $\sqrt{c}$ 游走.

根据文献[16]，当 $\{W_a'\}$ 为从 a 出发的反向随机 $\sqrt{c}$ 游走集合时，则

$$s(a,b)=\sum_{\substack{W_a'\in\{W_a'\}\\ W_b'\in\{W_b'\}_{W_a'}}}\boldsymbol{P}(b,W_a'),$$

其中，$\{W_a'\}$ 根据蒙特卡洛方法随机生成，而对于每个随机路径 W_a' 根据 ProbeSim 算法生成对应 $\{W_b'\}_{W_a'}$. 同理，对于 SimRank 的 Top-k 近似查询计算，根据 ProbeSim 算法，根据 $W_a'\in\{W_a'\}$ 中节点，根据正向 $\sqrt{c}$ 游走，计算 $\sqrt{c}$ 游走首先到达的 Top-k 节点.

如例 1 中若要计算节点“4”的 Top-k 相似查询，如图 6 所示. 首先生成一条 $\sqrt{c}-walk$，$W_4=4\leftarrow 2\leftarrow 1$，然后从节点 $W_4[2]=$“1”开始正向游走查询得到节点{“2”，“3”}.

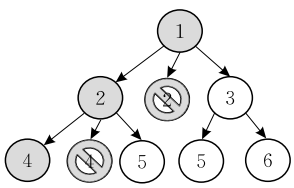

图 6 ProbeSim 算法

由于节点 2 在 W_4 中从节点 1 游走一步距离，所以删除，计算保存 $Score(\text{“3”},1)=\frac{\sqrt{c}}{|I(3)|}$，进一步探寻得到节点{“5”，“6”}，进而计算 $Score(\text{“5”},2)=Score(\text{“3”},1)\cdot\frac{\sqrt{c}}{|I(5)|}$，$Score(\text{“6”},2)=Score(\text{“3”},1)\cdot\frac{\sqrt{c}}{|I(6)|}$，$W_4$ 中节点 1 开始探寻结束；从节点 $W_4[1]=$“2”开始正向游走查询得到节点{“4”，“5”}，由于节点 4 在 W_4 中从节点 2 正向游走一步距离，因此删除节点 4，$Score(\text{“5”},1)=\frac{\sqrt{c}}{|I(5)|}$，因此此条随机游走探测到的 $s(5,4)=\frac{\sqrt{c}}{|I(3)|}\cdot\frac{\sqrt{c}}{|I(5)|}+\frac{\sqrt{c}}{|I(5)|}$.

ProbeSim 算法在进行 Top-k 查询时能够有效避免无效计算，提高计算效率；同时 ProbeSim 算法无需建立索引，直接生成$\sqrt{c}-walk$ 逐步计算，因此可以在动态变化图中直接查询；在大图中查询计算时间空间复杂度基本不变，与图的平均入度 d 以及阻尼系数 c 相关.

3.3.4　其他随机游走 SimRank 算法

除上文提及的 SimRank 计算随机游走算法，还有 SinglePair 算法[27]，以及 h-覆盖(h-go cover)算法[28]等随机游走算法.

其中 h-覆盖算法需要提前计算并缓存部分节点相似度，因此仍需要提前计算部分节点相似度，并且这种方法的时间空间最坏情况复杂度并没有给出有效证明.

另外，文献[35-36]提出 UniWalk 算法，与 ProbeSim 算法类似，不需要提前计算构建索引，但是 UniWalk 算法用于非有向图的 SimRank 计算，且 UniWalk 算法并非使用 ProbeSim 算法的$\sqrt{c}-walk$ 方式进行随机游走.

3.3.5　随机游走算法总结

根据随机游走的算法计算过程与特点，随机游走算法主要分为三类，基于不同索引结构的 SimRank 计算，基于不同抽样方式的 SimRank 计算与基于不同查询方式的 SimRank 计算. 通过压缩索引结构，降低预处理计算空间复杂度与查询阶段时间复杂度；通过改变建立索引时随机游走抽样方式，将随机游走步长信息通过抽样方式不同加入索引中，从而降低查询阶段时间复杂度；通过正向随机游走的查询方式，对单源查询只计算相似度大于 0 的节点对(相同步长下相遇的反向随机游走)，从而进一步降低查询计算时间复杂度.

由于基于随机游走 SimRank 计算只与索引中随机游走经过节点相关，与全图中其他节点相关性不大，在单源查询中，随机游走算法与全图规模相关度较小，因此随机游走算法的可扩展性较强，比如 ProbeSim 算法的实验中，图节点数量达到 40 M，这是前文中两类种算法所不能达到的量级；另外由于随机游走间相互独立，因此随机游走算法更容易进行并行计算.

但是由于随机游走算法使用蒙特卡洛模拟计算，随机游走抽样过程具有随机性，且随机游走步长会产生截断误差等，因此计算结果的稳定性与准确性较差；另外由于 SimRank 计算使用的为反向随机游走首次相遇概率，而随机游走算法多使用反向随机游走相遇概率，因此结果准确性较差，但是对于单源相似度查询结果影响不大，所以常用于单源查询，特别是 Top-k 查询. 本部分涉及的部分随机游走算法及其算法复杂度和扩展性如表 4 所示.

表 4　部分算法及复杂度分析(可扩展性顺序”好”，”较好”，”一般”，”较差”，”差”. 其中 δ 为计算结果误差大于 ε 的概率，t 为随机游走步长)

算法	问题	时间复杂度		空间复杂度	准确度	可扩展性
		预处理	查询			
FP-Sim	Top-k	$O\left(n\log\frac{1}{\varepsilon}\log\frac{n}{\delta}\Big/\varepsilon^2\right)$	$O\left(n\log\frac{1}{\varepsilon}\log\frac{n}{\delta}\Big/\varepsilon^2\right)$	$O\left(n\log\frac{1}{\varepsilon}\log\frac{n}{\delta}\Big/\varepsilon^2\right)$	ε	较好
TSF	Top-k	$O\left(n\log\frac{1}{\varepsilon}\Big/\varepsilon^2\right)$	$O\left(\left(\frac{n\log\frac{1}{\varepsilon}}{\varepsilon^2}\right)t^2+n\log n\right)$	$O\left(n\log\frac{1}{\varepsilon}\Big/\varepsilon^2\right)$	ε	好
SLING	Top-k	$O\left(\frac{m}{\varepsilon}+n\log\frac{n}{\delta}\Big/\varepsilon^2\right)$	$O\left(\frac{n}{\varepsilon}\right)$	$O\left(\frac{n}{\varepsilon}\right)$	ε	较好
READS	Top-k	$O\left(n\sqrt{\frac{\ln2-\ln\delta}{2\varepsilon^2}}\right)$	$O(n)$	$O\left(n\sqrt{\frac{\ln2-\ln\delta}{2\varepsilon^2}}\right)$	ε	好
TopSim	Top-k	$O(d^{2t})$	$O(k\log k)$	$O(d^{2t})$	ε	较好
ProbeSim	Top-k	$O\left(\frac{n}{\varepsilon^2}\log\frac{n}{\delta}\right)$	$O\left(\frac{n}{\varepsilon^2}\log\frac{n}{\delta}\right)$	$O(m)$	ε	好

4 SimRank 不同计算方法关系分析

4.1 迭代法与随机游走法之间关系

由式(4)及 $\boldsymbol{S}_0=\boldsymbol{I}$,令$(c\cdot\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{S}_{k-1}\boldsymbol{Q})\vee\boldsymbol{I}=(c\cdot\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{S}_{k-1}\boldsymbol{Q})_{\mathrm{off}}+\boldsymbol{I}$,其中$(c\cdot\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{S}_{k-1}\boldsymbol{Q})_{\mathrm{off}}$表示矩阵对角线元素全为 0,迭代法可表示为下式:

$$\boldsymbol{S}_k=c^k(\boldsymbol{Q}^{\mathrm{T}}\cdots(\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{Q})_{\mathrm{off}}\cdots\boldsymbol{Q})_{\mathrm{off}}+\cdots+c^0\boldsymbol{I},$$

式中 $c^0\boldsymbol{I}$ 表示图中所有节点经过 0 步随机游走首次相遇计算得到的 SimRank 值;$c^1(\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{Q})_{\mathrm{off}}$ 表示随机游走经过 0 步随机游走未相遇,而经过 1 步随机游走相遇计算得到的 SimRank;以此类推 $c^k(\boldsymbol{Q}^{\mathrm{T}}\cdots(\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{Q})_{\mathrm{off}}\cdots\boldsymbol{Q})_{\mathrm{off}}$ 为经过 0 次到 $k-1$ 次随机游走均未相遇,而经过 k 步随机游走相遇计算得到的 SimRank 值.

迭代法与随机游走算法不同在于,迭代计算中,经过 k 步随机游走计算得到的值为准确值,而随机游走算法只能通过蒙特卡洛算法抽样部分随机游走概率计算,迭代法需要缓存全部节点间相似度,计算空间时间复杂度与图的规模密切相关,而随机游走游走算法计算只与根据对应节点产生的随机游走索引有关,与图的规模相关性不高,因此迭代算法的可扩展性较差,而随机游走算法可扩展性较强.

4.2 迭代法与非迭代法之间关系

由非迭代法式(5)展开得到公式如下:

$$\boldsymbol{S}_k=(1-c)[c^k(\boldsymbol{Q}^{\mathrm{T}}\cdots(\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{Q})\cdots\boldsymbol{Q})+\cdots+c^0\boldsymbol{I}] \quad (6)$$

由上式看出,非迭代算法计算将所有随机游走相遇计算结果求和,然后乘以参数$(1-c)$进行修正得到 SimRank 结果. 同时,在文献[10]中证明,非迭代计算结果 $s'(a,b)$与迭代算法计算结果 $s(a,b)$满足下式:

$$1\leq\frac{s'(a,b)}{s(a,b)}\leq\frac{1}{1-c}.$$

例 2. 分别使用迭代法与非迭代法计算图 1 中节点相似度.

迭代法计算结果如下:

$$\boldsymbol{S}=\begin{bmatrix}1&0&0&0&0&0\\0&1&0.8&0&0&0\\0&0.8&1&0&0&0\\0&0&0&1&0.72&0.72\\0&0&0&0.72&1&0.72\\0&0&0&0.72&0.72&1\end{bmatrix}.$$

非迭代法计算结果如下:

$$\boldsymbol{S}=\begin{bmatrix}0.2&0&0&0&0&0\\0&0.36&0.16&0&0&0\\0&0.16&0.36&0&0&0\\0&0&0&0.408&0.208&0.208\\0&0&0&0.208&0.408&0.208\\0&0&0&0.208&0.208&0.488\end{bmatrix}.$$

由例 2 看出,迭代法与非迭代法计算结果并不完全相同,但是单源查询相似度顺序与迭代法相同. 根据迭代法与非迭代法计算原理,如果一个节点生成的随机游走,与其他节点生成的随机游走至多相遇一次,即每个节点只有一个入边,那么非迭代法与迭代法计算结果相同. 于是,我们可以得出以下定理.

定理. 若在有向图中,每个节点有且只有一个入边时,迭代算法与非迭代算法计算结果相同.

证明. 设图 $G(V,E)$邻接矩阵为 $\boldsymbol{P}$,若图中每个节点有且只有一个入边,则矩阵 $\boldsymbol{P}$ 每列只有一个元素为 1,所以$\boldsymbol{Q}=\boldsymbol{P}$,则$\boldsymbol{S}_k[i,i]=c\sum_{j=0}\boldsymbol{Q}^{\mathrm{T}}[i,j]\boldsymbol{S}_{k-1}\boldsymbol{Q}[i,j]+(1-c)$,且$\boldsymbol{S}_1[i,i]=c\sum_{j=0}\boldsymbol{Q}^{\mathrm{T}}[i,j]\boldsymbol{I}\boldsymbol{Q}[i,j]+(1-c)$,所以$\boldsymbol{S}_k[i,i]=c+(1-c)=1$,因此$(c\cdot\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{S}_{k-1}\boldsymbol{Q})\vee\boldsymbol{I}=(c\cdot\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{S}_{k-1}\boldsymbol{Q})+(1-c)\boldsymbol{I}$. 所以若在有向图中,每个节点有且只有一个入边时,迭代算法与非迭代算法计算结果相同. 证毕.

4.3 通过 G^2 图计算与通过原图计算关系

由 3.2.2 节定义,G^2 图的邻接归一化矩阵 $\mathbb{Q}=\boldsymbol{Q}\otimes\boldsymbol{Q}$,其中$\otimes$为 Kronecker 乘积. 因此原图 G 中任意两节点 SimRank 计算为,原图中任意两节点出发,经过 t 步反向随机游走首次相遇,则 $s=\mathbb{E}(c^t)$;而由于 G^2 中节点为 G 中节点对,图 G^2 中任意节点相似度为图 G 中节点对相似度,G^2 中任意节点相似度的计算,即从 G^2 中任意节点出发,反向随机游走,首次到达节点对$\langle i,i\rangle$的步长为 t,则节点相似度为 $s=\mathbb{E}(c^t)$.

文献[32,34]提出 PBiCG 算法与 FLP-SR 算法,通过 G^2 计算可以将 SimRank 计算过程优化为更简洁的线性系统,消除非线性操作符$\vee$;且由于计算过程中只需要进行迭代计算,因此可以将计算中稀疏矩阵进行压缩存储,同时,在 FLP-SR 算法中,只需要使用较大残差节点对的残差对计算结果中 SimRank 值进行更新,从而大大减少存储空间;并且由于计算使用的公式由原始SimRank定义表达式变换而得到,因此这种算法的精度能够得到保障. 由于 FLP-SR 算法更新过程只使用残差较大节点,因此计算时间空间复杂度大大降低,从而使计算可扩展性大大增强,FLP-SR 算法可以对 40 M 规模节

点的图进行 SimRank 计算查询.

4.4　不同算法之间特点

迭代法通过矩阵迭代计算，求解全体节点间 SimRank 值，其计算结果精度较高，如 Naïve 算法，但是由于迭代算法需要存储所有节点间相似度，因此空间复杂度较高，且计算过程涉及多次矩阵间乘法运算，因此时空复杂度较高，不适合较大规模图形的 SimRank 计算.

非迭代算法将 SimRank 非线性计算近似表示为线性系统，虽然非迭代计算 SimRank 值均有误差，但是对于相似度顺序影响不大，且避免反复迭代计算，但是由于存在矩阵求逆与分解运算，如 NI-SR 算法与 SimMat 算法，不适合较大规模图的计算，只适合中等规模图的计算；非迭代算法还可以化为用矩阵线性表示，中间结果可通过蒙特卡罗模拟随机游走进行求解，从而避免中间多次矩阵运算；或者针对部分节点集合间相似度计算，通过算法改变矩阵计算顺序，将“矩阵×矩阵”运算变为“矩阵×向量”运算，从而大大减少计算量，提高计算效率. 但是由于这些算法都需要存储转换矩阵，因此这些算法均不适合用于较大规模图的计算. 但是 FLP-SR 算法在更新 SimRank 计算结果时，只选取残差较大结果进行更新，从而大大降低计算时间空间复杂度，因此可以拓展到较大规模图的 SimRank 计算中，而其他 NI-SR 算法，只能进行中等规模图的 SimRank 计算.

随机游走算法通过建立随机游走索引，减少存储空间，加快计算速度，灵活性较强，可扩展性较好，且计算效率较高，如 FP-SR 算法、TFS 算法、SLING 算法与 READS 算法等，这些算法都需要经过两步，首先通过预计算建立索引，然后进行查询，一般预计算时间复杂度较高，且索引占用空间较大；Probe-Sim 算法等将计算过程缩减为直接“探测”进行 Top-k 查询，直接计算速度比其他建立索引算法计算速度稍慢，但是算法可扩展性大大提高，且由于没有索引建立过程，因此可以实现对动态图实时查询计算. 随机游走算法使用蒙特卡洛模拟计算，因此结果的稳定性稍差，但是对于较大规模的图以及动态更新的图算法可拓展性较好. 迭代算法与随机游走算法结合使用提高算法的灵活性与精确度，如 Par-SR 算法等.

5　SimRank 算法复杂度与可扩展性分析

5.1　迭代算法

迭代算法精度一般较高，且理论误差上限为 $\|\boldsymbol{S}_k-\boldsymbol{S}\|_{\max}\leq c^{k+1}$. 文献[7]首先提出 Naïve 算法为迭代算法，其根据式(4)，不断迭代，计算图中节点相似度. 由于每次计算中需要存储整个图的相似矩阵，因此 Naïve 算法空间复杂度为 $O(n^2)$，每次迭代涉及两次矩阵乘运算，因此计算时时间复杂度为 $O(kd^2n^2)$，其中，n 为图中节点数量，k 为迭代步数，d 为图中节点平均入度. 由于图节点相似度与邻近部分节点较高，因此计算过程可只选择每个节点随机游走 k 步所走到的节点，数量为 d_k，因此 Naïve 算法时间复杂度为 $O(knd_kd^2)$，空间复杂度为 $O(nd_k)$，计算过程中通过剪枝技术能够使时空复杂度进一步降低.

对于邻接矩阵比较稀疏的有向图，文献[17]提出一种优化方法 SOR-SR，对于稀疏邻接矩阵使用压缩稀疏行(CSR)技术，减少存储空间，加速计算，计算时间复杂度变为 $m\cdot n$；对于密度矩阵使用快速矩阵乘法运算，则计算时间复杂度为 n^γ，因此这种方法迭代时间复杂度为最坏情况为 $O(\min(km\cdot n, kn^\gamma))$，由于压缩存储，因此空间复杂度为 $O(m+n)$；针对迭代计算收敛较慢，使用连续过松弛算法加速收敛. 另外文献[18]提出“平方缓存”方法进一步提高收敛速度，并进一步提升到“u 次方缓存”.

SimRank 计算中由于节点间入边邻域经常有重叠，因此会有重复计算，同时文献[12]提出 Psum-SR 算法，减少中间计算中重复计算次数，提高计算效率. Psum-SR 算法将 Naïve 算法的时间复杂度减小为 $k\cdot\min(O(nd_r), O(n^3/\log_2 n))$，但是，需要存储中间计算结果，因此空间复杂度增加 $O(n)$. 文献[13]在文献[12]的基础上，提出 OIP-DMST 算法，通过改变部分求和顺序，先求解使用频率较高节点入边邻节点的部分求和，当对其他部分求和时，已经求解的这部分可以重用，从而进一步减少计算量，提高计算速度.

迭代算法具有精度较高等特点，但是迭代算法计算需要基于全体节点相似度进行迭代，中间过程需要存储全体节点间相似度，因此空间复杂度较高，不适合较大规模图的计算，可扩展性较差. 另外由于每次迭代需要基于全图信息，因此无法计算动态更新图中节点相似度.

5.2　非迭代算法

非迭代算法首先将 SimRank 计算基于式(5)表示为线性系统计算近似相似度值，然后通过线性代数低秩近似技术求解，或者将迭代过程通过线性展开进行求解.

NI-SR 过矩阵 SVD 分解,矩阵 Kronecker 乘积等运算进行预处理,预处理时间复杂度为 $O(r^4n^2)$,单点对查询的时间复杂度为 $O(r^4)$,空间复杂度为 $O(r^4+n^2r^2)$. 由于求逆过程与奇异值分解花费时间太多,以及预处理过程计算矩阵占用空间较大,因此 NI-SR 等规模以下图的 SimRank 计算. 但是 NI-SR 算法通过对动态图增量出的邻接矩阵分解,基于静态图中计算结果,能够快速计算 SimRank 值的变化.

SimMat 算法使用特征分解,基于西尔维斯特方程[20]求解. 预处理阶段需要进行矩阵分解等操作,时间复杂度为 $O(rn^2)$,空间复杂度为 $O(rn^2)$;SimMat 算法 Top-k 查询的时空复杂度最低为 $O(rn)$[19].

Par-Sim 算法通过种子生成模型,减少每次迭代计算的重复计算,并且通过剪枝算法,进一步减少计算复杂度. 单源查询时间复杂度为 $O(\min(km, d_{\max}^{2k}))$,空间复杂度为 $O(m+n)$.

由于非迭代方法一般需要进行矩阵分解与求逆运算,分为预计算阶段与查询阶段,预计算阶段产生的矩阵较大,因此非迭代方法的适用范围较小,一般仅适用于中小型规模图的查询计算. 通过矩阵运算的非迭代算法一般直接返回全体节点间相似度,而通过线性表示计算方式可直接进行单点对与单源查询计算. NI-SR 算法可进行动态图 SimRank 计算.

但是,近两年基于 G^2 图提出的非迭代算法,将 SimRank 计算中非线性操作符表示为线性系统,既保留了 SimRank 计算的准确度,又能够使用线性系统求解方式进行快速计算,如 PBiCG 算法[32];FLP-SR 算法[34]通过残差推进技术降低计算时间空间复杂度. 这些算法能够扩展到较大规模图的计算中进行全体节点对的 SimRank 计算,如 FLP-SR 可用于 40 M 节点数量图中 SimRank 计算.

5.3 随机游走算法

基于随机游走的方法根据定义 2,使用蒙特卡罗模拟方法随机抽取随机路径,计算有向图节点相似度. 基于随机游走方法主要区别为随机游走节点选择抽样方式不同,如$\sqrt{c}-walk$ 等,以及生成的随机游走建立索引的方式不同,如生成指纹图. 基于随机游走方法 SimRank 值与真实值误差为 ε,抽样失败概率为 δ,即 $P\{|s(a,b)-s_{\text{randomwalk}}(a,b)|\geq\varepsilon\}\leq\delta$.

文献[15]提出的 FP-SR 算法预处理过程建立 r_g 组指纹树,时空复杂度为 $O(nr_g)$,其中 r_g 由计算精度决定,预处理时空复杂度为 $O\left(n\log\frac{1}{\varepsilon}\log\frac{n}{\delta}/\varepsilon^2\right)$[16];查询阶段对节点 a 进行 Top-k 查询,只需查询每个子图中含有 a 节点分支. 此算法单点对查询时间复杂度为 $O\left(\log\frac{1}{\varepsilon}\log\frac{n}{\delta}/\varepsilon^2\right)$[16]. 由于此算法提前建立索引,且索引难以更改,因此难以用于动态图计算.

文献[21]提出的 TSF 算法,预处理阶段生成 r_g 个单路图,因此时空复杂度为 $O(nr_g)$,r_g 由计算精度确定为 $O\left(n\log\frac{1}{\varepsilon}/\varepsilon\right)$;对于 Top-$k$ 查询,查询时间复杂度为 $O(r_gr_qt^2+n\log k)$,$r_gr_q=O\left(n\log\frac{1}{\varepsilon}/\varepsilon\right)$. 另外此算法针对图的动态变化从而动态更新抽样出的单路图,因此可以用于动态图计算.

文献[16]使用的随机游走抽样过程使用了新的抽样方法 SLING. SLING 算法通过生成$\sqrt{c}$ 游走方式计算 SimRank. 单源节点对时间复杂度为 $O\left(\frac{n}{\varepsilon}\right)$,空间复杂度为 $O\left(\frac{n}{\varepsilon}\right)$,预处理时间为 $O\left(\frac{m}{\varepsilon}+n\log\frac{n}{\delta}/\varepsilon^2\right)$. 查询阶段算法避免额外计算,因此 SLING 算法查询结果稳定性、速度以及准确性均比传统抽样计算结果高,但是对于动态图需要重新构建索引.

文献[23]提出的 ProbeSim 算法,使用$\sqrt{c}$游走进行探测计算,直接进行 Top-k 查询,没有预计算处理过程,单次查询过程需要多次抽样计算. 但此算法没有建立预处理随机游走索引,查询时间复杂度为 $O\left(\frac{n}{\varepsilon^2}\log\frac{n}{\delta}\right)$,空间复杂度为 $O(m)$. 由于未建立索引,因此此算法可用于动态图的实时查询,且查询时间与图的更新几乎无关,算法可扩展性较强.

除以上随机游走算法,Monte-Carlo Single-Pair[24]算法结合非迭代算法迭代运算展开,通过蒙特卡洛模拟中间矩阵计算结果,从而直接求解单点对相似度;TopSim[25]算法通过维持两条随机游走路径,计算候选节点集合,从而加速 Top-k 查询计算等.

随机游走算法首先建立随机游走索引,查询阶段通过计算索引中随机游走相遇概率与步长节点相似度,避免不相似节点间计算,从而提高计算效率. 随机游走法具有较高的可扩展性,能够快速计算动态图相似度,如 TFS、ProbeSim 等.

6　SimRank 计算总结与展望

6.1　SimRank 计算总结

从 Jeh 与 Widom 首次提出 SimRank 模型到现在，人们对 SimRank 计算研究已经有 16 年，研究人员提出了各种精确及近似计算查询的方法，根据计算方式可将这些算法分为三类：迭代法、非迭代法与随机游走法.

传统的迭代法计算精度较高，结果稳定性好，但是计算时空复杂度高，不适合大规模图计算；传统非迭代方法将 SimRank 计算进行线性近似表示，预处理阶段通过求解线性系统进行计算，查询阶段根据计算得到的全体相似度直接查询，适用于中等规模图相似查询，但是近两年基于 G^2 提出的 SimRank 计算，既能保留计算精度，又能够提高计算扩展性，可以在 40 M 节点数的大规模图中进行全体节点对 SimRank 计算；随机游走算法，预处理阶段通过不同的抽样方式及索引存储结构建立索引，查询阶段通过计算首次相遇步长与概率计算，随机游走算法的计算结果稳定性较差，预处理建立索引占用空间较大，但是查询计算效率较高，查询过程时空复杂度较低，可扩展性好.

近几年，对较大规模图的单点对/单源计算研究越来越多，而传统的迭代法与非迭代法常用于全体节点间相似度计算，文献[14]通过研究迭代算法与随机游走算法，将其应用于单点对/单源查询计算，并通过改变计算顺序，减少计算量；对于随机游走算法需要预处理，因此难以满足当下大规模图的计算的时效性要求，文献[23]不需要预处理过程，从而大大提高计算的时效性与可拓展性；另外，使用路径相似类算法[21,23,29]，针对单源查询问题或 Top-k 查询，可以大大减少中间结点相似度计算量，提高计算效率；对于动态变化的图，随机游走算法通过免除索引建立过程[23]，适应动态图近似 Top-k 查询的时效性要求；对于有多邻居节点间 SimRank 计算相似度偏低，以及不同随机游走相遇但相遇路径长度不同的节点间相似度低等问题，文献[32]对 SimRank 模型定义进行了改进. 文献[34]提出的非迭代算法，将图的全体节点对计算规模提升到 10^7 量级.

6.2　SimRank 计算主要挑战与展望

如图 7 所示，近几年 SimRank 计算中图的规模越来越大，由十年前节点数为 10^5 量级达到了现在的 10^7 量级；同时可以看出非迭代算法与随机游走算法的计算效率较高，其中非迭代算法一般针对全体节点对求解问题，而随机游走算法一般针对单源查询或单点对查询效率较高. 因此，SimRank 计算的挑战主要集中于以下几个方面：大规模图的 SimRank 快速计算；SimRank 值的精确计算；以及动态图中 SimRank 值的快速更新计算.

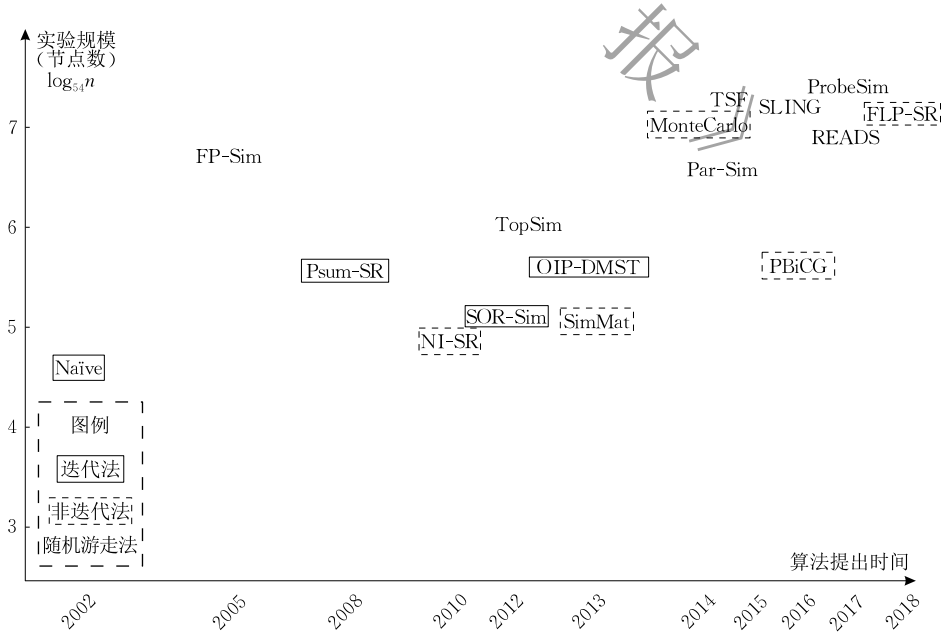

图 7　文章涉及部分算法发布时间及实验规模关系图

由于数据规模越来越大，SimRank 计算问题规模提升了两到三个量级，因此 SimRank 计算的效率要求越来越高，由上图可以看出由于传统的迭代算法本身固有的缺点，近些年对其的研究越来越少，而多集中于随机游走算法与非迭代算法. 由于非迭代算法多使用矩阵运算等，因此可以使用 GPU 等新

硬件进行加速计算,如文献[26]对 NI-SR 算法的改进;或者使用分布式计算,如文献[38,39]等使用 Spark 等分布式系统来提升系统的扩展性与计算效率.

由于传统 SimRank 算法[11,19]计算中使用$(1-c)\boldsymbol{I}$来替代对角修正矩阵$\boldsymbol{D}$,从而消除非线性操作符,但是这种算法计算结果并非 SimRank 定义值,而近两年文章[32,34]基于G^2图重新改写 SimRank 计算定义式,既消除非线性操作符,保持 SimRank 定义的一致性与准确性,同时还大大提升全体节点对的计算效率,因此可以将其他 SimRank 查询问题基于G^2图进行进一步研究.

由于常见信息网络随时间会产生动态变化,SimRank 计算的时效性要求越来越高,特别是大规模社交网络的 SimRank 计算,因此基于动态变化图的 SimRank 快速更新算法越来越多,如 ProbeSim 算法[23]与 UniWalk 算法[35-36]等算法,由于没有构建索引过程,因而可以直接对动态图进行 Top-k 查询;而文献[34]的 FLP-SR 算法可以基于已有计算结果对动态变化后的图的全体节点对进行快速更新,文献[40]提出一种算法可以针对边的动态增加,基于原有计算结果对全体节点对相似矩阵进行快速更新.

因此 SimRank 计算未来主要挑战为大规模图中 SimRank 的高效计算,大规模图中 SimRank 的准确计算,以及动态大规模图中 SimRank 的快速更新计算.这需要对 SimRank 定义表达式进行进一步改进,消除非线性项,借助分布式与 GPU 等新硬件平台,对 SimRank 快速计算,同时更需要设计够兼容 SimRank 快速计算以及在动态变化图中能快速更新的新算法.

参 考 文 献

[1] Nguyen P, Tomeo P, Di Noia T, Di Sciascio E. An evaluation of SimRank and Personalized PageRank to build a recommender system for the Web of Data//Proceedings of the 24th International Conference on World Wide Web. Florence, Italy, 2015: 1477-1482

[2] Abbassi Z, Mirrokni V S. A recommender system based on local random walks and spectral methods//Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis. California, USA, 2007: 102-108

[3] Liben-Nowell D, Kleinberg J. The link-prediction problem for social networks. Journal of the American Society for Information Science and Technology, 2007, 58(7): 1019-1031

[4] Benczúr A A, Csalogány K, Sarlós T. Link-based similarity search to fight Web spam//Proceedings of the 2nd International Workshop on Adversarial Information Retrieval on the Web. Seattle, USA, 2006: 9-16

[5] Zhou Yang, Cheng Hong, Yu J X. Graph clustering based on structural/attribute similarities. Proceedings of the VLDB Endowment, 2009, 2(1): 718-729

[6] Rothe S, Schutze H. CoSimRank: A flexible & efficient graph-theoretic similarity measure//Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics. Baltimore, USA, 2014: 1392-1402

[7] Jeh G, Widom J. SimRank: A measure of structural-context similarity//Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Edmonton, Canada, 2002: 538-543

[8] Xi Wensi, Fox E A, Fan Weiguo, et al. SimFusion: Measuring similarity using unified relationship matrix//Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Salvador, Brazil, 2005: 130-137

[9] Zhao Peixiang, Han Jiawei, Sun Yizhou. P-Rank: A comprehensive structural similarity measure over information networks//Proceedings of the 18th ACM Conference on Information and Knowledge Management. Hong Kong, China, 2009: 553-562

[10] Zhang Zhipeng, Shao Yingxia, Cui Bin, Zhang Ce. An experimental evaluation of SimRank-based similarity search algorithms. Proceedings of the VLDB Endowment, 2017, 10(5): 601-612

[11] Li Cuiping, Han Jiawei, He Guoming, et al. Fast computation of SimRank for static and dynamic information networks//Proceedings of the 13th International Conference on Extending Database Technology. New York, USA, 2010: 465-476

[12] Lizorkin D, Velikhov P, Grinev M, Turdakov D. Accuracy estimate and optimization techniques for SimRank computation. Proceedings of the VLDB Endowment, 2008, 1(1): 422-433

[13] Yu Weiren, Lin Xuemin, Zhang Wenjie. Towards efficient SimRank computation on large networks//Proceedings of the 2013 IEEE 29th International Conference on Data Engineering (ICDE). Brisbane, Australia, 2013: 601-612

[14] Yu Weiren, McCann J A. Efficient partial-pairs SimRank search on large networks. Proceedings of the VLDB Endowment, 2015, 8(5): 569-580

[15] Fogaras D, Rácz B. Scaling link-based similarity search//Proceedings of the 14th International Conference on World Wide Web. New York, USA, 2005: 641-650

[16] Tian Boyu, Xiao Xiaokui. SLING: A near-optimal index structure for SimRank//Proceedings of the 2016 International Conference on Management of Data. New York, USA, 2016: 1859-1874

[17] Yu Weiren, Zhang Wenjie, Lin Xuemin, et al. A space and time efficient algorithm for SimRank computation. World Wide Web, 2012, 15(3): 327-353

[18] Yu Wei-Ren. Research on Optimization Techniques for Pervasive Structural Similarity Computation on Large Scale Networks[Ph. D. dissertation]. Donghua University, 2012: 1-147(in Chinese)

(俞唯仁. 普适的结构相似度在大规模网络中的计算优化技术研究[博士学位论文]. 东华大学, 2012: 1-147)

[19] Fujiwara Y, Nakatsuji M, Shiokawa H, Onizuka M. Efficient search algorithm for SimRank//Proceedings of the 29th International Conference on Data Engineering (ICDE). Brisbane, Australia, 2013: 589-600

[20] Benner P. Factorized solution of Sylvester equations with applications in control. sign (H), 2004, 1: 2. https://www.math. ucsd. edu/~ helton/MTNSHISTORY/CONTENTS/2004LEUVEN/CDROM/papers/59. pdf

[21] Shao Yingxia, Cui Bin, Chen Lei, et al. An efficient similarity search framework for SimRank over large dynamic graphs. Proceedings of the VLDB Endowment, 2015, 8(8): 838-849

[22] Jiang M, Fu A W C, Wong R C W. READS: A random walk approach for efficient and accurate dynamic SimRank. Proceedings of the VLDB Endowment, 2017, 10(9): 937-948

[23] Liu Yu, Zheng Bolong, He Xiaodong, et al. ProbeSim: Scalable single-source and top-*k* SimRank computations on dynamic graphs. Proceedings of the VLDB Endowment, 2017, 11(1): 14-26

[24] Kusumoto M, Maehara T, Kawarabayashi K. Scalable similarity search for SimRank//Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. Snowbird, USA, 2014: 325-336

[25] Lee P, Lakshmanan L V S, Yu J X. On top-*k* structural similarity search//Proceedings of the 2012 IEEE 28th International Conference on Data Engineering. Washington, USA, 2012: 774-785

[26] He Guoming, Li Cuiping, Chen Hong, et al. Using graphics processors for high performance SimRank computation. IEEE Transactions on Knowledge and Data Engineering, 2012, 24(9): 1711-1725

[27] Li Pei, Liu Hongyan, Yu J X, et al. Fast single-pair SimRank computation//Proceedings of the 2010 SIAM International Conference on Data Mining. Columbus, USA, 2010: 571-582

[28] Zheng Weiguo, Zou Lei, Feng Yansong, et al. Efficient SimRank-based similarity join over large graphs. Proceedings of the VLDB Endowment, 2013, 6(7): 493-504

[29] Tao Wenbo, Yu Minghe, Li Guoliang. Efficient top-*k* SimRank-based similarity join. Proceedings of the VLDB Endowment, 2014, 8(3): 317-328

[30] He Guoming, Feng Haijun, Li Cuiping, Chen Hong. Parallel SimRank computation on large graphs with iterative aggregation //Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Washington, USA, 2010: 543-552

[31] Yu Weiren, Wang Fan. Fast exact CoSimRank search on evolving and static graphs//Proceedings of the 2018 World Wide Web Conference on World Wide Web. Lyon, France, 2018: 599-608

[32] Lu Juan, Gong Zhiguo, Lin Xuemin. A novel and fast SimRank algorithm. IEEE Transactions on Knowledge and Data Engineering, 2017, 29(3): 572-585

[33] Maehara T, Kusumoto M, Kawarabayashi K. Scalable SimRank join algorithm//Proceedings of the 2015 IEEE 31st International Conference on Data Engineering. Seoul, South Korea, 2015: 603-614

[34] Wang Yue, Lian Xiang, Chen Lei. Efficient SimRank tracking in dynamic graphs//Proceedings of the 2018 IEEE 34th International Conference on Data Engineering (ICDE). Paris, France, 2018: 545-556

[35] Luo Xiongcai, Gao Jun, Zhou Chang, Yu J X. UniWalk: Unidirectional random walk based scalable SimRank computation over large graph//Proceedings of the 2017 IEEE 33rd International Conference on Data Engineering (ICDE). San Diego, USA, 2017: 325-336

[36] Song Junshuai, Luo Xiongcai, Gao Jun, et al. UniWalk: Unidirectional random walk based scalable SimRank computation over large graph. IEEE Transactions on Knowledge and Data Engineering, 2018, 30(5): 992-1006

[37] Lizorkin D, Velikhov P, Grinev M, Turdakov D. Accuracy estimate and optimization techniques for SimRank computation. The International Journal on Very Large Data Bases, 2010, 19(1): 45-66

[38] Li Zhenguo, Fang Yixiang, Liu Qin, et al. Walking in the cloud: Parallel SimRank at scale. Proceedings of the VLDB Endowment, 2015, 9(1): 24-35

[39] Gao Xingkun, Bao Nianyuan, Liu Jie, et al. Scalable single-source SimRank computation for large graphs//Proceedings of the 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS). Wuhan, China, 2016: 1083-1091

[40] Yu Weiren, Lin Xuemin, Zhang Wenjie, McCann J A. Dynamical SimRank search on time-varying networks. The International Journal on Very Large Data Bases, 2018, 27(1): 79-104

ZHANG Liang-Fu, Ph. D. candidate. His research interests include data mining, machine learning.

LI Cui-Ping, Ph. D. professor, Ph. D. supervisor. Her research interests include data mining, recommendation systems, information network analysis, stream data management, etc.

CHEN Hong, Ph. D., professor, Ph. D. supervisor. Her research interests include big data management and privacy protection, data management and data analysis based on new hardware, data warehousing and data mining, etc.

Background

SimRank was first proposed by Glen Jeh and Jennifer Widom in 2002. The core idea of SimRank similarity is that, if two objects are referenced by their similar objects (that is, they have similar adjacent edge structures), then the two objects are similar. In recent years, it has attracted extensive attention in the field of information retrieval, and has been successfully used in web page ranking, collaborative filtering, outlier detection, network graph clustering, approximate query processing, and so on. Over the past decades there're many algorithms that efficiently calculate or approximately calculate SimRank scores being proposed. According to the characteristics of the algorithm, it can be divided into the following three categories, iterative algorithm, non-iterative algorithm and random walk algorithm. Meanwhile, there are two categories of SimRank computing according to the problem, Single Source (Pair) Query and All (Partial) Pair Query problems. Different algorithms are suitable for different problems. These algorithms are based on two definitions of SimRank, and in order to facilitate the design of the new algorithms, we analyze the relationship of the random walk method and the other two methods, summarize the time and space complexity of these algorithms, as well as the advantages and disadvantages of these algorithms. At last, we make some prospect of SimRank computation and conclude the challenge of the SimRank computation.

This work was supported in part by the National Key R&D Program of China under Grant No. 2018YFB1004401, in part by the National Natural Science Foundation of China under Grant Nos. 61772537, 61772536, 61702522 and 61532021.