图随机游走引擎综述

邹鸿志 [1)]

[1)] 华中科技大学计算机科学与技术学院

摘　要　许多应用程序需要学习、挖掘、分析和可视化大型图形, 这些图通常太大，无法使用传统的图处理技术有效地处理。而通过图的采样和随机游走可以大大减少原始图的大小，可以通过捕获所需的图属性来帮助学习、挖掘、分析和可视化大型图。因此作为图数据分析和机器学习的一种工具，图随机游走获得了巨大的欢迎。目前，随机游走算法都是独立实现的，存在显著的性能和可伸缩性问题，特别是复杂游走策略的动态性以及随机游走读取时的 I/O 低效问题，严重影响了随机游走实现的效率。目前主流的用于图处理方面的框架系统，通常采用以顶点或边为中心的模型，对单个图操作进行了高度优化。然而它们都只关注于传统的图查询操作，而没有考虑到随机游走算法的工作负载，因此许多专门针对随机游走算法进行优化的框架引擎逐渐被开发。其中 KnightKing 是第一个被提出的随机游走引擎，采用分布式架构，使用拒绝采样技术使得运行效率比传统图计算系统得到 4 个数量级的提升。GraphWalker 采用一种能异步游走更新的 I/O 状态感知模型，使得单机运行效率达到了与采用分布式架构的 KnightKing 在 8 台机器上运行效率相同的效果。C-SAW 是第一个基于使用 GPU 加速的框架，其提出的基于 GPU 的二分区域搜索和多 GPU 调度技术也取得了很好的效果。ThunderRW 是一个基于内存的随机游走框架，使用步进交错技术，使得在大的工作负载时，有效减少内存访问延迟，显著地将 CPU 流水线的延迟时间从 73.1% 降低到 15.0%。这些随机游走框架的提出，提供用户以随机游走为中心的视角，便利了开发使用过程。

关键词　图计算引擎；随机游走；KnightKing；GraphWalker；C-SAW；ThunderRW

Title Research On Graph Random Walk Engine

HongZhi Zou[1)]

[1)]College of Computer Science and Technology, Huazhong University of Science and Technology

Abstract

Many applications need to learn, mine, analyze and visualize large graphs, which are usually too large to be processed effectively using traditional graph processing techniques. Therefore, graph random walk has been greatly welcomed. The mainstream framework system only focus on the traditional graph query operation without considering the workload of random walk algorithm. So many framework engines specially optimized for random walk algorithm have been developed. KnightKing is the first random walk engine proposed. It uses reject sampling technology to improve the operation efficiency by 4 orders of magnitude compared with the traditional graph computing system. GraphWalker adopts an I / O state aware model, so that the single machine operation efficiency is the same as that of KnightKing with distributed architecture on 8 machines. C-SAW is the first framework based on GPU acceleration, and it has also achieved good results. ThunderRW is a random walk framework based on memory and significantly reduce the delay time of CPU pipeline from 73.1% to 15.0%. These random walk frameworks provide users with a random walk centered perspective and facilitate the development process.

Keywords　Graph Engine；Random Walk；KnightKing；GraphWalker；C-SAW;ThunderRW

1 引言

图随机游走算法是一种为了从图中的实体间提取路径信息的高效的、应用广泛的图处理工具。它为许多重要的图度量、排序和图嵌入算法奠定了基础，比如 PageRank、SimRank、DeepWalk 和 Node2Vec 等，这些算法可以独立工作，也可以作为机器学习任务的预处理步骤。它们服务于不同的应用，如节点/边缘分类、社区检测、链路预测、图像处理、语言建模、知识发现、相似度测量和推荐。近年来，学术界和工业界也有越来越多的工作关注随机游走，根据微软学术的统计结果，2018 年就有大约 1700 篇学术论文是关于随机游走，工业界的很多大公司比如 Facebook、谷歌、微软、阿里巴巴和腾讯等也都使用了随机游走相关的技术。

图随机游走算法输入为一张图，开始时会同时启动多个随机游走过程，每个随机游走具有一定的行走长度，然后利用这些随机游走的访问模式进行图数据分析。一个随机游走首先从一个初始顶点开始，随机选择当前顶点的一个邻居顶点并跳转到该顶点，重复上述过程直至满足预设的终止条件，比如随机游走达到一定的步长或者随机游走在每一步有一定概率终止。很多基于随机游走的应用程序常常需要同时运行大量的随机游走过程，从而保证计算的准确性。

基于对图随机游走算法应用的分析，我们发现基于随机游走的图计算过程具有如下特征：1）对图数据访问的随机性更强，2）并发随机游走的数量可能很大，3）随机游走的步长可能非常长。首先由于图数据通过边数据错综复杂的关联关系，图计算场景的数据访问本身往往就具有很强的随机性。而随机游走在每一步的转移过程中随机挑选当前顶点的邻居顶点的跳转方式，更是加剧了其对图数据访问的随机性。其次，基于随机游走的采样中会有独立采样的方式，需要从图中同时出发很多条随机游走，然后在这些随机游走收敛之后分别采样一个样本。在这种情况下为保障采样精确性，往往会同时启动大量的随机游走。最后，基于随机游走的图采样都需要在随机游走到达收敛时再进行采样，而随机游走的收敛往往也需要非常多的步数，因此带来了超长随机游走的应用场景。

然而目前已经出现的用于传统图处理方面的图计算引擎，不能作为提供通用随机游走计算框架的系统，来实现高效和可伸缩的图随机游走。因此，使用者往往每次都要开发自己的随机游走实现，这样大大增加了冗余劳动，也不能提供优良的性能。同时对于用户来说，目前流行的图框架处理系统都专注于沿边更新顶点状态，而随机游走这种以步行点为中心的算法会变的违反直觉。另外许多最先进的图引擎的许多系统优化，如 2 维图分区和类似 GAS 的执行，并不适合随机游走计算模型，甚至可能适得其反，降低性能。因此为了解决如上问题，提出了许多专门用于处理随机游走过程的计算引擎。其中最先提出的 KnightKing，采用了分布式架构，主要针对复杂的动态随机游走，提出了一种拒绝采样的策略，将一个一维的采样过程转化为一个二维的采样过程，消除了对当前顶点所有边的采样，从算法层面上大大加快了采样效率。相比于传统的图处理引擎，在执行速度方面带来了高达 4 个数量级的改进。后来针对在大型图上进行随机游走时非常低的 I/O 效率问题，提出了 GraphWalker，该模型利用每个随机遍历的状态，优先将遍历最多的图存储块从磁盘加载到内存，从而提高 I/O 利用率，同时提出一种以随机游走为中心的异步更新策略，大大提升随机游走更新速度。在运行效果上 GraphWalker 在单机上的处理速度可以达到 KnightKing 使用 8 台机器的速度。随后提出的 C-SAW 实现了一个在 GPU 上加速采样和随机游走的框架，在不同图的实验结果上都取得了很好的效果。而由新加坡大学提出的 ThunderRW，针对不规则内存访问问题，开发了步进交错技术，通过切换不同随机遍历查询的执行来减少内存访问延迟，在性能表现方面通过步进交错技术，显著地将 CPU 流水线延迟从 73.1% 降低到 15.0%。这些用于随机游走过程的框架引擎极大优化和遍历了随机游走的计算过程。

2 原理与优势

2.1 随机游走

2.1.1 随机游走基本分类

基于随机游走的算法遵循一个共同的框架：给定一个图，一定数量的随机游走过程分别从一个给定的起点出发，然后根据给定的概率分布，选择当前驻留顶点的一个邻边，并移动到于此邻边相连的邻点，简单来说不同随机游走算法的关键变化在于邻边的选择步骤。

从边被选择的相对概率上来看，随机游走算法可以分为无偏和有偏。无偏是指出边转移概率不依赖于它们的权值或者其他性质，而有偏是指在随机游走过程中对邻边的选择进行加权。如果边转移概率在整个过程中保持不变，则为静态随机游走，否则为动态随机游走，其中的转移概率的确定涉及到随机游走的状态，它在游走过程中不断变化。因此，

在动态随机游走过程中，需要在每一步重新计算这些概率，而不是预先计算所有的边转移概率。
除了转移概率定义外，不同的随机游走算法可能会采用不同的终止策略。常见的策略包括在给定的步数上停止行走 (导致行走序列的长度一致)，或者在每一步上以给定的概率结束行走。

2.1.2 典型随机游走算法

PPR： PPR 是 PageRank 的一个更复杂的版本。与通常使用幂迭代计算的一般 PageRank 不同，对于大图 PPR 需要耗费大量时间和空间才能有效地计算。因此，把基于随机游走而生成的行走序列作为 PPR 个性化推荐的方式成为一种近似计算方案，这是一个有偏静态的随机游走算法，用于模拟 Web 浏览中的个人偏好。PPR 算法的目标是要计算所有节点相对于用户 u 的相关度。从节点 v 对应的点开始游走，每到一个节点都以 1-d 的概率停止游走并从 v 重新开始，或者以 d 的概率继续游走，从当前节点指向的节点中按照均匀分布随机选择一个节点往下游走。这样经过很多轮游走之后，每个顶点被访问到的概率也会收敛趋于稳定，这时就可以通过每个点的概率来进行排名。

DeepWalk： DeepWalk 是一种在机器学习中广泛使用的图嵌入技术。它基于 SkipGram 模型，对于每个顶点，它以目标长度开始指定数量的随机游走查询来生成用于训练输入的点及其邻域。最初的 DeepWalk 是无偏的，而最近的许多工作将其扩展到考虑边权值的情况，从而变成静态有偏随机游走。

Node2Vec： 作为一种二阶随机游走的图嵌入技术，Node2Vec 在图嵌入方面也深受欢迎。不同于 DeepWalk，它的边转移概率取决于游走中访问的上一个节点。对于点 v 而言，其上一步访问节点为 u，则边 $e(v,v')$ 的转移概率 $p(e(v,v'))$ 如公式 (1) 所示，其中边的转移概率由上一次访问的节点 v' 与 u 之间的距离 $dist(v',u)$ 决定，而 a 和 b 是两个控制随机游走行为的超参数。Node2Vec 是动态的，因为转移概率依赖于查询的状态，同时通过 $p(e)$ 与边权值w_e 相乘，可以将边的权重考虑进来。

$$p(e(v,v')) = \begin{cases} \frac{1}{a}, & if \quad dist(v',u)=0 \\ 1, & if \quad dist(v',u)=1 \\ \frac{1}{b}, & if \quad dist(v',u)=2 \end{cases} \tag{1}$$

2.1.3 采样方法

从离散概率分布 P=$\{p_0, p_1, \cdots, p_{n-1}\}$ 中抽样的过程就是在基于 P 的 $\{0, 1, \cdots, n-1\}$ 之间选择一个元素 h, 即 $P[h=i]=p_i$。本文主要介绍三总采样技术，分别是翻转变换采样、别名采样，因为它们十分高效并且使用广泛。

翻转变换采样（ITS）是静态随机游走中的一种常见的采样技术，如图 1 所示。ITS 的初始阶段是计算 P 的累计分布概率，表达式为 $P'=\{p'_i=\sum_{j=0}^{i} p_j\}$，其中 $0 \le i < d_v$。随后生成阶段首先生成一个实数 x，范围为 $[0, p'_{d_v-1})$，然后使用二分查找找到最小的索引 i，使得 $x < p'_i$，并且最终选择 $E_v[i]$。整个过程中初始化阶段时间复杂度为 $O(d_v)$，生成阶段复杂度为 $O(\log\ d_v)$。

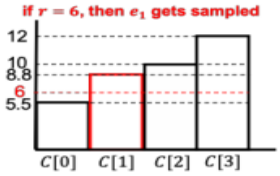

图 1 ITS

别名采样（ALIAS）初始化阶段需要构建两个表：概率表 H 和别名表 A，它们两者都有d_v 个值，如图 2 所示。H[i] 和 A[i] 分别代表 H 和 A 的第 i 个值。给定 $0 \le i < d_v$,A[i] 是包含范围为 $\{0, 1, \cdots, d_v-1\}$ 中一个或两个元素的桶，其中元素由 A[i].first 和 A[i].second 来表示，H[i] 是选择 A[i].first 的概率。如果 A[i] 只有一个元素，那么 A[i].second 为空并且 H[i] 值为 1。生成阶段首先产生一个范围在 [0,d_v) 之间的整数 x，然后检索出对应的 H[x] 和 A[x]。接下来生成一个范围在 [0, 1) 之间的实数 y, 如果 $y < H$[x], 那么选择 $e(v,A$[x].first) 作为采样成功的边，否则选择 $e(v,A$[x].second) 作为采样成功的边。算法初始化阶段时间复杂度为 $O(d_v)$，生成阶段的时间复杂度为 $O(1)$，空间复杂度为 $O(d_v)$。

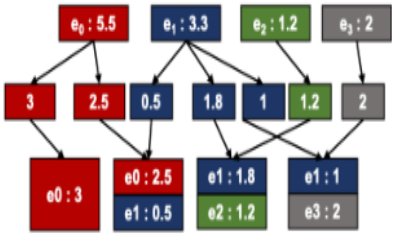

图 2　别名采样

2.2　随机游走处理引擎

2.2.1　传统图处理框架

在不同的计算环境中有许多通用的图计算框架，比如（1）单机：GraphChi，Ligra，Graphene 和 GraphSoft；（2）GPU：Medusa，CuSha 和 Gunrock；(3) 分布式环境：Pregel，GraphLab。这些框架通常采用顶点或边为中心的模型，并对单个图操作进行了高度优化。然而，它们都只关注传统的图查询操作，如 BFS 和 SSSP, 而没有考虑随机游走算法的工作负载，这推动了专门针对随机游走进行优化的框架引擎的开发。

2.2.2　随机游走引擎

与传统的图计算框架从图数据的角度抽象计算相比，现有的随机游走框架采用了以游走步进为中心的模型，将每个查询视为并行任务。其中 KnightKing 是一个分布式架构的框架引擎，它采用 BSP 模型，在每次迭代中为所有查询移动一步，直到所有查询完成。C-SAW 是一个基于 GPU 加速的框架引擎，它也使用了 BSP 模型。GraphWalker 是一个在单机上 I/O 高效的框架引擎。ThunderRW 是基于解决内存访问问题而实现的框架。这些随机游走引擎的出现，极大方便了用户效率。

3　研究进展

3.1　KnightKing

KnightKing 是第一个提出的通用随机游走运算的框架引擎，是一个分布式的随机游走引擎。它提供了以游走步行者为中心的视图，使用自定义的 API 来定义边的转移概率，同时处理其他随机游走的基础设置。与图计算引擎类似，KnightKing 隐藏了图形分区、顶点分配、节点间通信和负载均衡等系统细节。因此，它提供了一个直观的“像游走步行者一样思考”的视图，同时用户可以添加可选的优化。

3.1.1　统一转移概率定义

KnightKing 定义了一个决定边转移概率的一般框架，它适用于已知的随机游走算法。对于一个随机游走过程 w 而言，当前驻留节点为 v，则与之相连的边的转移概率定义为静态概率部分P_s，动态概率部分P_d 和扩展概率部分P_e，更具体的来说，转移概率$P(e)$=$P_s(e)$·$P_d(e,v,w)$·$P_e(v,w)$, 其中状态 w 包含了必要的历史信息，比如以前访问过的 n 个顶点。

在这样的框架下，原始简单的算法就变成了有偏、高阶算法的特殊情况，比如一个无偏、静态的算法的P_s 和P_d 的值都定义为 1。与P_s 和P_d 的定义无关，当随机游走到达一定长度的步长或者满足指定的终止概率后，P_e 变为 0, 同时当没有外边存在或者有正转移概率的边存在时，遍历终止。

3.1.2　拒绝采样算法

在处理静态随机游走时，KnithtKing 使用了已经存在的别名采样方法进行边的采样，而对于动态随机游走边的采样，由于不能高效的进行采样前的与计算，KnightKing 提出了一种拒绝采样的方法。其过程为先允许一条边被采样，然后检查被采样的边是否满足采样概率。这个方式与之前的采样方式看起来区别很小，但是其避免了每步游走时需要扫描当前顶点所有边的昂贵开销。在具有合理的概率包络范围时，采样成功的概率很大，每一步具有 $O(1)$ 的时间复杂度去获得采样成功的边。 以如

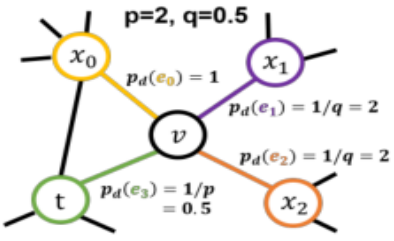

图 3　采样图

图 3 所示的动态随机游走 Node2Vec 举例，类似于别名采样，首先根据给定的静态概率P_s 进行一个 1 维的概率采样，获取候选边。然后根据候选边的动态概率P_d 决定候选边是否采样成功，如果成功，游

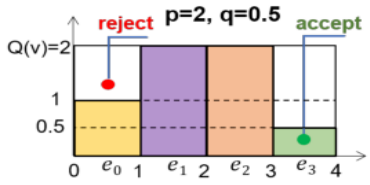

图 4　点 v 出的拒绝采样

走到下一个节点，否则重新进行一次采样，因此每个单独采样的时间复杂度为 $O(1)$ 或者 $O(n)$。完成的采样效率取决于采样试验的平均次数。直观上来说，每次试验的结果对应于 “有效面积” 的比值，即所有条形物的组合面积除以整个矩形面积，对于一条边进行采样的平均试验次数 E 有如下公式 (2)：

$$E = \frac{Q(v) \cdot \sum_{e \in E_v} P_s(e)}{\sum_{e \in E_v} P_s(e) \cdot P_d(e)} \tag{2}$$

从公式上可以看出平均采样次数与边的度数量无关，因此使用合适的 $Q(v)$，可以使得即使有巨大的顶点度，采样次数也可以很小，产生显著的性能提升。

3.1.3　提供的 API 框架

KnightKing 为用户提供了直观的 API 来指定现有的或创新的随机游走算法。

转移概率定义是用户需要提供给 KnightKing 的核心信息，以实现自定义随机游走算法。KnightKing 提供了两个相关的 API 用于静态和动态转移概率的定义，分别为 edgeStaticComp 和 edgeDynamicComp。

用户填充这些函数来重写系统默认的分配方式，其中默认设定这些概率都为 1。通过 edgeStaticCompy，用户提供静态转移概率的定义，通常作为边的权值。由于该定义不涉及随机游走的状态，因此可以预先设定。基于这个定义，KnightKing 在初始化期间相应地执行准备工作，比如构建别名表。edgeDynamicComp 提供涉及无法预先计算的随机游走状态的动态计算定义。对于高阶算法，其中动态边转移概率计算涉及到其他顶点，用户调用另一个接口 postStateQuery，提交进行随机游走状态的查询。

初始化和终止条件的定义，是在初始化时，KnightKing 设置随机游走的数量，然后通过使用提供的 API 可以指定特定的起始位置或起始位置分布。当两种方式都没有指定时，开始位置是使用 KnightKing 的默认策略确定的。类似的，KnightKing 为用户提供设置终止条件的 API，用于设定自定义的终止条件。

随机游走状态由 KnightKing 自动执行默认的状态维护方式，例如自动更新当前驻留的顶点和行走的步数等信息。此外，用户可以使用提供的 API 执行自己的初始化和更新自定义游走状态。

3.1.4　小结

KnightKing 是第一个通用的分布式图随机行走引擎。它提供了一个直观的以步行者为中心的计算模型，以支持随机步行者算法的简单规范。同时提出了一个统一的边缘转移概率定义，适用于流行的已知算法，以及新的基于拒绝的采样方案，极大地降低了昂贵的高阶随机漫步算法的代价。KnightKing 也提供了一个供用户设置使用的 API 框架，极大的方便了用户的使用过程。通过对 KnightKing 的系统进行实验评估，结果表明无论当前顶点出边的数量如何，都可以实现复杂度接近 o(1) 的不精确采样，而不损失精度。

3.2　GraphWalker

GraphWalker 是为了解决 I/O 效率问题，从而有效地支持快速和可扩展的随机游走过程。该系统可以高效地处理由数十亿个顶点和数千亿条边组成的非常庞大的磁盘驻留图，而且还可以并发地运行数百亿条、数千步长的随机游走。实验结果表明，在运行大量随机行走时，与原有最好的单机随机游走图处理系统 DrunkardMob 相比，GraphWalker 的处理速度快一个数量级；与分布式随机游走图处理系统 KnightKing 相比，GraphWalker 在一台机器上的处理速度可以达到其 8 台机器上的速度；与性能最好的通用单机图处理系统 Graphene 和 GraGSoft 相比，GraphWalker 在支持随机游走任务时，速度提升也非常明显，在最好的情况下，速度也快一个数量级。

3.2.1　状态感知的图加载

图数据的组织和切分。GraphWalker 使用广泛应用的压缩稀疏行 (compressed Sparse Row, CSR) 格式管理图形数据该格式将顶点的外邻居顺序存储在磁盘的 acsr 文件中，并使用索引文件记录每个顶

点的起始位置在 CSR 文件中。GraphWalker 根据顶点 id 将图划分为块，根据 id 的升序排列依次添加顶点和它们的出边进入块中，直到块中的数据容量达到预设的容量大小，然后创造一个新的块。图 5 显示了一个示例图，图 6 显示了图 5 中实例图的数据布局。

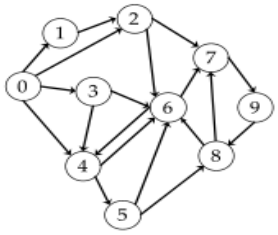

图 5　示例图

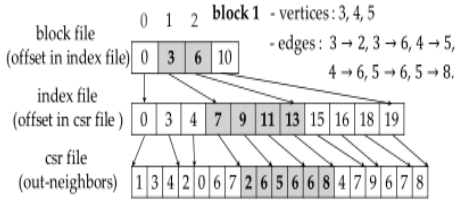

图 6　数据组织布局。
图以 CSR 格式存储，块文件中记录块分区范围。

此外，这种轻量级的图数据组织方式降低了每个子图的存储成本，从而降低了加载图的时间成本。因为 GraphWalker 通过简单地读取索引文件来记录每个块的起始顶点来对图进行分区，它还可以灵活地为不同的应用程序调整块的大小。同时还存在一种设置子图块尺寸的权衡策略。使用较小的块可以避免加载更多的数据，这些数据并不需要更新随机游走，而使用较大的块可以在每个子图加载中更新更多的随机游走。此外，不同的分析任务需要不同的步行尺度，因此偏好不同的块大小。具有少量遍数的轻量级任务喜欢较小的块大小，因为在此设置下可以提高 I/O 利用率。相比之下，具有大量行走的重量级任务更喜欢大的块大小，因为大的块大小可以增加行走的更新速率。基于此，GraphWalker 使用了一种基于经验的分析方法，设置子图块的大小为 $2^{\lg R+2}$MB, 式中 R 为随机游走的总次数。例如，在运行 10 亿次随机游走的情况下，默认的块大小是 2 GB，通常小于一台商品机器的内存容量，因此很容易将一个图块保存在内存中。

图加载和块缓存。GraphWalker 在预处理阶段转换图形格式并划分图形块。在运行随机遍历的阶段，GraphWalker 选择一个图块，并根据遍历的状态将其加载到内存中，特别是，它将加载包含最多遍历的块，在完成对加载的图形块的分析后，它然后选择另一个块以同样的方式加载。为了减轻块大小的影响并提高缓存效率，GraphWalker 还支持块缓存，方法是开发一种行走感知的缓存方案，将多个块保存在内存中，其基本原理是具有更多随机游走访问的块更有可能在将来被访问。因此，使用块缓存的图的加载过程如图 7 所示。

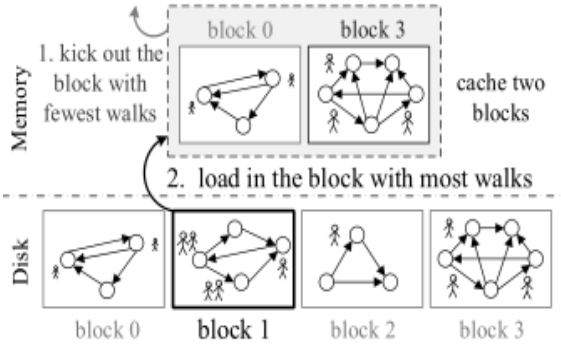

图 7　使用块缓存的状态感知图形加载。

首先根据状态感知模型选择一个候选块，为了加载这个块，需要检查它是否缓存在内存中。如果它已经在内存中，那么直接访问内存来执行分析。否则，将从磁盘加载它，并且如果缓存已满，还将取出内存中包含最少遍历的块，缓存在内存中的最大块数量取决于可用的内存大小。实验表明对子图分块敏感方案总是优于传统的缓存方案。

3.2.2　异步随机游走更新策略

在基于迭代的系统中，加载一个图块后，加载的子图中的每次遍历只走一步，这导致了非常低的遍历更新速率。事实上，在进行了一步随机游走后许多游走点仍然位于现在子图中，所以在此子图中能更新更多的步数。为了提高 I/O 效率，有些工作采用了加载后的数据重入，有些工作使用交叉迭代值传播来形式化对加载数据的重用，以便为后续迭代提供同步保证。但是，这种重新进入的方案可能会引起局部离散问题。，许多游走能够游走一步当图块第一次加载的时候，但是随着数据重载的数量

增大, 许多游走可能达到子图的边界, 只有少数留在子图, 它们需要多个从新载入才能完成。

为了进一步提高 I/O 利用率和遍历更新速率, GraphWalker 采用了异步遍历更新策略，允许每次遍历保持更新，直到到达加载的图块的边界。在完成一次遍历之后，选择另一次遍历进行处理，直到处理完当前图形块中的所有遍历，然后，根据上面描述的状态感知模型加载另一个图块。图 8 显示了在同一个图块中处理两个遍历的示例。为了加速计算，GraphWalker 还使用多线程并行更新步数，在异步遍历更新模型中，完全避免了无用的顶点访问，并消除了局部流浪器问题。然而，状态意识模型可能

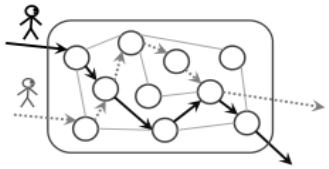

图 8　异步遍历并行更新。

会导致全局滞后问题。也就是说 GraphWalker 可以快速地完成大多数遍历，但是要完成剩下的几个遍历需要很长时间。为了解决查问题 GraphWalker 引入了一种概率方法来处理状态感知的图加载过程。这样做是为了让掉队的随机游走有机会多走几步，这样他们就能跟上大多数随机游走的步伐。

3.2.3　小结

GraphWalker 是一个高效的 I/O 系统，支持快速和可伸缩的随机游走在单个机器上的大型图。GraphWalker 仔细地管理图数据和遍历索引，并通过使用状态感知的图加载和异步遍历更新优化 I/O 效率。在原型机上的实验结果表明，GraphWalker 的性能优于目前最先进的单机系统，也达到了与集群机上的分布式图系统相当的性能。

3.3　C-SAW

C-SAW 是第一个基于 GPU 的框架，支持广泛的采样和随机游走算法。C-SAW 也采用了 BSP 模型，为了充分利用多核体系结构中的并行计算能力, C-SAW 在计算中采用了 ITS 采样。对于包括无偏、静态和动态在内的所有随机游走类型，C-SAW 首先对相邻边的转移概率进行前缀和计算，然后选择一条将要游走的边。C-SAW 提出了一个通用的框架，它允许终端用户轻松地表达大量的抽样和随机漫步算法, 同时采用新颖的技术实现高效的 GPU 采样，并行化了 GPU 上的顶点选择，为顶点碰撞迁移提供了高效的算法和系统优化。C-SAW 还提出了对采样和随机游走的异步设计，它优化了图形的数据传输效率，超过 GPU 内存容量，进一步扩展到多个 GPU。

3.3.1　采样框架

C-SAW 把在 GPU 上的采样和随机游走过程简化成表现丰富的 API 和有很好表现的框架。简化意味着终端用户可以在不知道 GPU 编程语法的情况下进行编程。表现丰富要求 C-SAW 不仅要支持已知的采样算法，并且也要为新出现的算法做好准备。C-SAW 使用户通过参数和 API 选择两种方式进行应用。基于参数的选项只需要最终用户的选择，因此很简单，比如决定选择的边界顶点的数量和邻居数量。基于 API 的形式为用户提供了更多的应用表达能力。C-SAW 提供了三个供用户定义的 API 函数，分别为 VERTEXBIAS(Vertex v), EDGEBIAS(Edge e),UPDATE(Edge e)。

3.3.2　GPU 采样优化

C-SAW 算法的核心是从顶点池中选择一个子集，其采用 ITS 方法进行 GPU 顶点选择，它允许用灵活和动态的偏差来计算转移概率，并且它显示了对 GPU 执行更友好的更常规的控制流程。为了并行化顶点选择循环，C-SAW 为每个顶点选择分配一个线程，以最大化并行性。对于每个循环迭代，生成一个随机数来选择一个顶点。然而，这就产生了一个关键的挑战，即不同的线程可能会选择相同的顶点，即选择冲突。为了解决上述选择冲突，提出了两种解决方案: 二分区域搜索和基于位图的冲突检测。其中二分区域搜索既继承了重复采样和更新采样的优点，又避免了它们的缺点。也就是说，与更新采样相比，它不需要昂贵的 CTPS 更新，而与重复采样相比，它大大提高了成功选择的机会。同时为了解决争用问题, C-SAW 使用跨界位图, 跨步位图将相邻顶点的位分散到不同的 8 位变量上，如图 9 所示。与使用同一个 8 位变量的前 5 位来指示连续位图中所有顶点的状态不同，跨步位图将它们分散到两个变量中以减少冲突。

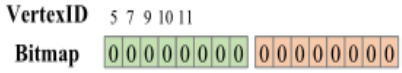

图 9　通过连续位图采样邻居节点。

图 10　通过跨步位图采样邻居节点。

3.3.3　多 GPU 的 C-SAW

随着资源数量的不断增长，工作负载将饱和甚至超出一个 GPU 的能力。在这种情况下，将 C-SAW 扩展到多个 GPU 将有助于提高采样性能。由于各种抽样实例是相互独立的，C-SAW 只是将所有的采样分成几个不想连的组，每一组中包含相同数量的采样实例，不相连的组的数量与 GPU 的数量相同，每个 GPU 负责一个采样组。在采样过程中，每个 GPU 将执行如图 10 所示的相同任务，并且不需要 GPU 之间的通信。

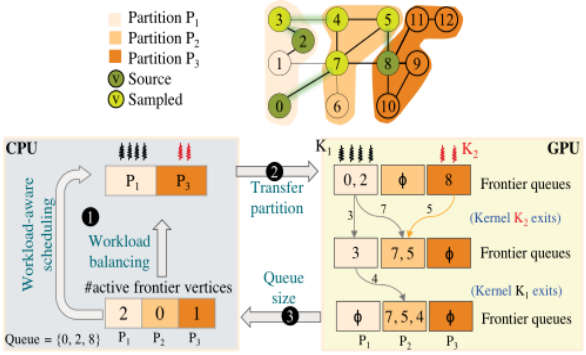

图 11　工作负载感知的图分区调度。

C-SAW 通过简单地将一个相邻的、相等范围的顶点和它们的邻居列表分配给一个分区来划分图，通过工作量感知来区分调度，由于多个采样实例彼此独立，这种灵活性赋予了 C-SAW 基于图分区和工作负载，动态调度各种分区的自由。C-SAW 跟踪每个分区中的边界顶点的数量，以确定哪个分区将提供更多的工作负载，如图 11 中的黑点 1 所示，基于这个计数，还将线程块分配给每个 GPU 内核，并在下一段中描述基于工作负载平衡的线程块。随后，工作量较大的分区会更早地转移到 GPU 上，并先采样如图 11 中黑点 2 所示。在计算方面，C-SAW 将一个 GPU 内核和一个 CUDA 流分别分配给一个活动分区，以重叠不同活动分区的数据传输和采样。并行分区采样完成后，计算每个指向队列中的顶点数，决定下一步哪个分区应该转移给 GPU 进行采样，如图 11 中黑点 3 所示。

3.3.4　小结

C-SAW 一种新颖的、通用的、优化的 GPU 图采样框架，它支持广泛的采样和随机漫步算法。C-SAW 中引入了新的以偏差为中心的框架、二分区域搜索和工作负载感知的 GPU 和多 GPU 调度，从实验效果看，C-SAW 取得了很好的效果。

3.4　ThunderRW

ThunderRW 是一个通用、高效的内存随机游走框架，采用以步进为中心的规划模型，将计算从步行者移动一步的局部角度抽象出来，用户通过在用户定义函数中“像步行者一样思考”来实现随机游走算法。框架将用户定义函数应用到每个查询，并将查询的一个步骤视为一个任务单元，从而并行执行查询。此外，ThunderRW 提供了多种采样方法，用户可以根据工作负载的特点选择合适的采样方法。在以步进为中心的规划模型的基础上，ThunderRW 提出了步进交错技术来解决软件预取过程中由于不规律的内存访问而导致的缓存延迟问题。由于现代的 CPU 可以同时处理多个内存访问请求，步进交错的核心思想是通过发出多个未完成的内存访问来隐藏内存访问延迟，这利用了不同随机游走查询之间的内存级别并行性。实验显示 ThunderRW 也表现出优良的性能效果。

3.4.1　系统框架

以步进为中心的模型。随机游走算法建立在多个游走查询的基础上，而不是单个的查询。尽管随机游走算法内部查询的并行性有限，但由于每个随机游走查询都可以单独执行，因此存在大量查询并行性。所以 ThunderRW 以步进为中心的模型从查询的角度抽象了随机游走算法，以利用查询间的并行性。ThunderRW 从一个查询 Q 进行一步移动的局部视角，对计算进行建模。在以步进为中心的模型中，用户通过“像步行者一样思考”来开发 RW 算法，他们专注于定义函数设定转移概率并且更新在查询 Q 每一步中的状态，方便了用户定义面向步

进的函数。

提供的 API。与 C-SAW 类似，ThunderRW 提供了两种 api，包括超参数和用户定义函数。用户分为两个步骤来开发随机游走算法。首先，通过超参数 walker_type 和 sampling_method 分别设置随机游走类型和采样方法。然后定义权重 Weight 和 Update 函数，其中 Weight 函数定义了一条边被选中的相对几率，Update 函数用于修改查询中被选择边的状态，如果返回结果是 true，则框架终止查询 Q，否则 Q 继续在图上游走。ThunderRW 将用户定义函数应用于随机游走查询，并基于随机游走类型和选择的采样方法并行地评估查询。因此，用户可以很容易地使用 ThunderRW 实现定制的随机游走算法，这大大减少了工程工作量。

并行性。ThunderRW 采用静态调度方式，保持随机游走之间的负载均衡。ThunderRW 把每一个线程视为一个随机游走过程，并且把查询 Q 任务平均的分给每个随机游走过程。

3.4.2 步进交错技术

ThunderRW 使用软件预取技术来加速 ThunderRW 的内存计算，然而对于一个查询 Q 的单独一步来说，没有足够的计算工作负载来隐藏内存访问延迟，因为查询 Q 的每一步之间相互独立，因此 ThunderSW 通过交替执行不同的查询步骤来隐藏内存访问延迟。当给定一组操作序列后，将它们分解为多个阶段，这样，一个阶段的计算将使用前一个阶段生成的数据，并在必要时检索后续阶段的数据。然后同时运行一组查询，一旦一个查询 Q 的一个阶段完成了，马上转移到一个组中的其他查询上，随后当其他查询完成后，继续执行查询 Q。通过这种方式，ThunderSW 可以在单个查询中隐藏内存访问延迟，并保持 CPU 繁忙，这种方法称为步进交错技术。图 12 给出了一个步骤分为四个阶段的例子。如果循序渐进地执行查询，那么 cpu 经常会因为内存访问而暂停。即使使用预取，阶段的计算也不能隐藏内存访问延迟。相反，步骤交错通过交替执行不同查询的步骤来隐藏内存访问延迟。如果一个组包含 k 个随机游走查询，假设没一个随机游走查询运行了相同步数阶段，并且每一阶段的开销W_c 也相同。假设有 m 个运行的阶段需要进行内存访问，$\overline{m}$ 个不需要,W_L 代表内存开销。那么顺序查询移动一步的代价开销为W_0=k($(m+\overline{m})W_C+mW_L$)。若W_S 代表转换开销，则步进交错的开销为W_1=$k((m+\overline{m})(W_C+W_S)+m(\max(W_L-kW_S-(k-1)W_C,0)))$，其中最后一项计算的是步骤交错是否隐藏了内存访问延迟。因此对于 k 个随机游走查询中的一步使用步进交错技术获得的收益能够用公式 (3) 来表示，其中W_hide=max(W_L-kW_S-(k-1)W_C,0)。

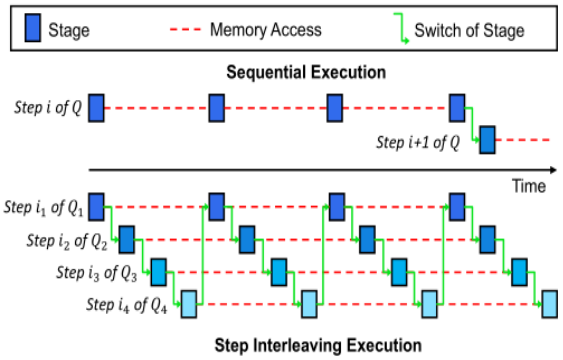

图 12 顺序运行与步进交错运行。

$$
\begin{aligned}
W_{gain} &= (W_0 - W_1)/k \\
&= mW_L - (m+\overline{m})W_S - mW_{hide}
\end{aligned}
\tag{3}
$$

从公式 (3) 中可以看出步进交错需要一个有效的开关机制来减少交错转换时的开销W_S，并且要有足够的工作负载来重叠内存访问延迟W_{hide}。

3.4.3 运行环调整

任务环的大小 k 和查询环的大小k' 确定线程中同时执行的查询的组大小，从而分别控制执行非循环阶段和循环阶段的内存级别并行性。根据公式 (2) 可以通过增加 k 来减少W_{hide} 从而提高性能，然而 k 被硬件所限制。现代的 CPU 可以发出有限数量的未完成的内存请求，L1 数据缓存大小只有几十 kb，设置 k 为一个大的数，可以在使用前清楚数据。ThunderRW 通过预执行一些查询来调整环的大小，从每个目标长度为 10 的顶点开始随机游走查询，并将随机游走类型设置为静态。通过分别使用原始采样和别名从采样的方法，从 $1, 2, \cdots, 512, 1024$ 范围内选择一个最佳的值k^*。接下来固定 k 的值为k^*，然后在 1,2,$\cdots$,k^* 范围内调整k' 的值，最终得到一

个最佳的值分别用于 ITS 和拒绝采样。

3.4.4 小结

ThunderRW 一个高效的内存随机游走引擎，用户可以轻松地实现定制的随机游走算法。ThunderRW 设计了一个以步长为中心的模型，将计算从查询移动一步的局部视图中抽象出来，在此基础上，提出了步进交错技术，通过交替执行多个查询来隐藏内存访问延迟。实验结果显示 ThunderRW 的性能比目前最先进的 RW 框架高出一个数量级，并且步进交错将内存限制从 73.1% 降低到 15.0%。

4 总结与展望

作为图数据分析和机器学习的一种工具，图随机游走逐渐受到更多的欢迎，专门针对其进行优化的计算框架引擎也逐渐提出，本文介绍了其中四种引擎。其中 KnightKing 是第一个被提出的随机游走引擎，采用分布式架构和拒绝采样的技术使动态随机游走算法的采样运行效率得到巨大提升。GraphWalker 采用一种能异步游走更新的 I/O 状态感知模型，使得随机游走在单机上运行也能实现相当的性能。C-SAW 是第一个基于使用 GPU 加速的框架，采用的基于 GPU 的二分区域搜索和多 GPU 调度技术也取得了很好的效果。ThunderRW 是一个基于内存的随机游走框架，使用步进交错技术，使得在大的工作负载时，有效减少内存访问延迟。以上提出的随机游走计算框架极大便利了用户使用效率。

随着多任务、多场景的随机游走的发展，相应的随机框架会继续有相应的改进。（1）面向新型存储设备的存储管理优化扩展。近年来新型的存储设备如 NVMe SSD、NVRAM 等，在访问特性上与传统设备具有差异，许多已有的优化策略在这些设备上收益减少。因此研究更适应这些新型设备的随机游走数据访问和存储优化方法成为发展可能。（2）多任务并发场景下的随机游走调度优化拓展。很多基于随机游走的图分析方法可能并发的从不同方面去分析同一个底层图数据，各任务间会竞争 CPU、I/O 及内存等资源，并相互干扰缓存，所以处理效率下降。因此可以考虑多个基于随机游走的应用并发处理场景，优化多任务并发图处理性能。综上，随机游走引擎以后会具有更多的场景和更丰富的发展。

参 考 文 献

参考文献

[1] Yang K , Zhang M X , Chen K , et al. KnightKing: a fast distributed graph random walk engine[C]// the 27th ACM Symposium. ACM, 2019.

[2] Wang R, Li Y, Xie H, et al. Graphwalker: An i/o-efficient and resource-friendly graph analytic system for fast and scalable random walks[C]//2020 USENIX Annual Technical Conference (USENIXATC 20). 2020: 559-571.

[3] Pandey S, Li L, Hoisie A, et al. C-SAW: A framework for graph sampling and random walk on GPUs[C]//SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 2020: 1-15.

[4] Sun S, Chen Y, Lu S, et al. ThunderRW: An in-memory graph random walk engine[J]. 2021.