图随机游走系统综述

叶楚玥[1)]

[1)]华中科技大学计算机科学与技术学院，武汉市 430074

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系统逐渐被开发。其中KnightKing 是一个分布式架构的系统，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 加速的系统，它也使用了BSP 模型。GraphWalker 是一个在单机上I/O 高效的系统。ThunderRW 是基于解决内存访问问题而实现的系统。FlashMob 是旨在提高缓存利用效率的系统。这些随机游走系统的提出，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 随机游走；KnightKing；GraphWalker；C-SAW；ThunderRW；FlashMob

中图法分类号 TP **DOI号:**

A Survey of Random Walk Systems

Chuyue Ye[1)]

[1)]Department of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074

Abstract

Many applications need to learn, mine, analyze, and visualize large graphs that are often too large to be effectively processed using traditional graph processing techniques. Therefore, as a tool of graph data analysis and machine learning, graph random walk has gained great popularity. The current mainstream systems for graph processing usually adopt a vertice/edge-centric model, which is highly optimized for a single graph operation. However, they only focus on the traditional graph query operation, and do not consider the workload of random walk algorithm. Therefore, many systems optimized for random walk algorithm are gradually developed. KnightKing is a distributed architecture system that adopts the BSP model and moves one step for all queries in each iteration until all queries are completed. C-SAW is a GPU-based accelerated system that also uses the BSP model. GraphWalker is an I/O efficient system on a single machine. ThunderRW is a system based on solving memory access problems. FlashMob is a system designed to make cache utilization more efficient. The introduction of these random walk systems provides users with a random walk centric perspective, which facilitates the development and use of random walk algorithms.

Keywords Random Walk; KnightKing; GraphWalker; C-SAW; ThunderRW; FlashMob

收稿日期： - - ；最终修改稿收到日期： - - . 本课题得到华科基金(No.1)资助.叶楚玥(通信作者)，女，1998年生，硕士在读，是计算机学会(CCF)会员(N2467G),主要研究领域为图神经网络、图计算.E-mail: 308591412@qq.com. 第1作者手机号码: 18811573858, E-mail: 308591412@qq.com

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，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系统，不能作为支持通用随机游走计算的系统，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不直观。另外许多最先进的图计算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。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系统。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%。随后，为了有效地利用现代CPU 内存层次结构，FlashMob被提出，它打破了人们普遍认为的随机游走固有的随机性和图的偏斜性使得大多数内存访问都是随机的，通过仔细的分区、重排和操作批处理，挖掘了大量的空间和时间局部性，提高了缓存和内存带宽的利用率，它还利用一个经典的组合优化问题作决策，以实现准确而高效的数据/任务划分，实验表明，该系统比现有最快的系统实现了一个数量级的性能改进。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 背景

2.1 图随机游走基础知识

给定一个图$G = (V, E)$和一个起始顶点$u \in V$，随机行走者w进行如下的操作。u的每一个邻居v有一个转移概率，该概率决定了v会被选为下一个顶点的可能性。典型地，转移概率$p(v|u)$ 只取决于u，这种情叫一阶随机游走。在高阶随机游走中，概率以$p(v|u, t, s, \ldots)$ 的形式指定, 其中, s, t是当前行走中u的前驱，计算出边的转移概率时要考虑w的行走历史。w根据这个概率采样一条边进行，并重复这一操作直到满足特定的终止条件。

终止可以是确定性的(在进行了给定的步数之后)，也可以是随机的(行走者在每一步以固定的概率退出)。

基于随机游走的算法都遵循上述框架，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。

由于随机游走在节点嵌入应用中被广泛使用，在这里也给出节点嵌入的简单介绍。节点嵌入是图学习的基本组成部分。给定一个图$G=(V,E)$和一个维度$d<<|V|$，节点嵌入是把每个节点$v\in V$用一个d维向量$Emb(v)$代表，使图G的结构信息得以保留。直观地说，如果一个节点对u和v有“相似”的邻域，那么它们的嵌入$Emb(u)$和$Emb(v)$也彼此接近。反之，具有不同邻域的两个节点的嵌入会相距更远。通过在正节点对集合P和负节点对集合N上训练深度学习模型来学习节点嵌入，使得$(u,v)\in P$的节点嵌入彼此更接近，而$(u,v)\in N$的节点嵌入则彼此相距较远。

通常，N是通过随机选取节点对来构造的(考虑到现实世界图的稀疏性，随机选取的节点对不太可能是连通的)，而P是通过随机游走算法来构造的。广泛使用的节点嵌入算法有DeepWalk和Node2Vec，它们在使用随机游走衡量邻域相似性的方式上有所不同，使用不同的转移概率定义。更具体地说，DeepWalk执行一阶均匀随机游走，而Node2Vec则是一种带有超参数的二阶算法。通常，这两种算法都采用默认参数，从图中的每个节点开始，执行10次长度分别为40和80的随机游走。以下是对这两种算法的详细介绍。

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。

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

2.2 随机游走系统设计

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, GraphSoft; (2)GPU: Medusa, CuSha, 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 随机游走框架

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。FlashMob 是旨在提高缓存利用效率的框架。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 是第一个通用随机游走运算的框架，是一个分布式的随机游走引擎。它提供了以行走者为中心的视图，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“像行走者一样思考”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

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, 同时当没有出边或者正转移概率的边存在时，遍历终止。

3.1.2 拒绝采样算法

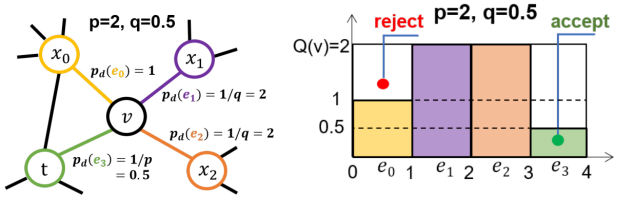

图1 拒绝采样示例图

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1 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$O(n)$。完成采样的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有如公式：

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

从公式上可以看出平均采样次数与边的度数无关，因此使用合适的$Q(v)$，可以使得即使有很大的顶点度，采样次数也可以很小，从而得到显著的性能提升。

3.1.3 工作流和编程模型

与传统图引擎在迭代中协调多个顶点(沿着多个边)的更新类似，KnightKing 拥有一个迭代计算模型，可以同时协调多个行走者的行动。与顶点和边一样，行走者也会根据当前驻留顶点的分配分配给每个节点/线程。当然，它与传统图引擎的明显区别在于，它是沿采样边的概率游走，而不是沿所有活动边的确定性更新传播。一个更微妙的不同之处在于每个迭代的执行都使用了分布式随机游走引擎。虽然图引擎可以通过一轮的顶点到顶点消息来推送/拉取更新和执行顶点状态更新，但在处理高阶遍历时，KnightKing 需要包含两轮消息传递。因为每个行走者可能需要基于最近访问的顶点执行边选择(例如node2vec, 检查前一点t是否匹配后一点x)。在分布式环境下,顶点和边跨节点分区,需要传递消息来发送这样的查询并收集它们的结果。

为了促进这种分布式查询操作的高效批处理和协调，KnightKing扮演了一个类似于邮局的角色，其中行走者根据他们的本地采样候选者提交查询消息，并将其发送到涉及动态检查的顶点。所有这些查询都是根据顶点到节点的分配来交付的，所有节点都并行处理从所有行走者接收到的查询。然后，另一轮消息传递将共同返回结果，查询的行走者将一起检索结果。以下是KnightKing 迭代的步骤:

1．行走者生成拒绝采样的候选边并进行初步筛选。

2．在必要时，行走者根据采样结果向顶点发出行走者状态查询。

3．所有节点处理状态查询并返回结果。

4．行走者检索状态查询结果。

5．行走者决定采样结果，如果成功就移动。

注意，上面描述的是KnightKing 支持的最一般的随机游走情况。用更少的复杂算法，通常步骤可以跳过。例如，在静态或一阶随机游走的情况下，由于在局部采样时不需要涉及其他顶点，因此省略了步骤2-4。对于这样的算法，所有的行走者都可以锁步移动: 在每次迭代中，行走者执行他们的采样和(局部)检查，直到一条边被成功采样(或行走终止)。

在这些情况下，所有活跃的行走者都沿着他们的行走顺序走相同的步骤。但是，对于像node2vec这样的高阶算法，行走者之间的迭代是通过两轮从行走者到顶点的查询消息传递实现同步的。成功采样的幸运者将继续向前走一步，而不那么幸运的人则停留在当前的顶点等待下一次迭代。这样，行走者可以以不同的速度前进，潜在地产生掉队者。

KnightKing 为用户提供了直观的API 来实现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。

随机游走状态是由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。

3.2 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。

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(CSR)格式管理图数据，该格式将顶点的出边顺序存储在磁盘的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2 显示了一个示例图，图3 显示了图2 中示例图的数据组织布局。 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。

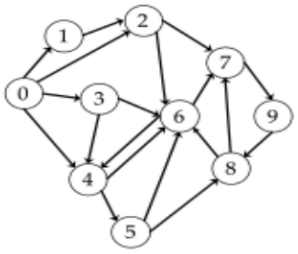

图2　示例图

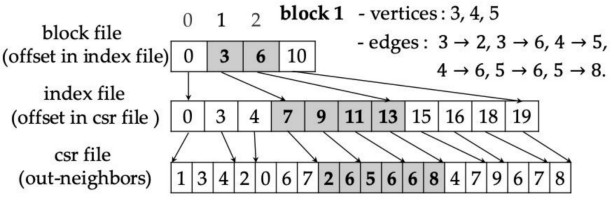

图3　数据结构

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^{lgR+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。

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。

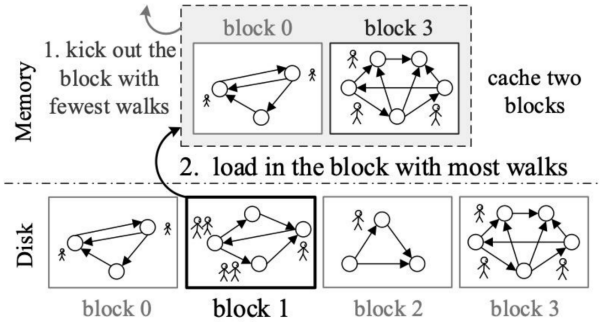

图4 使用块缓存的状态感知图加载

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。

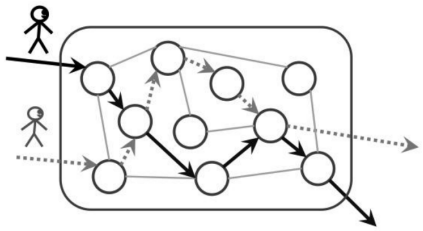

图5 异步游走并行更新

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5 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，在异步游走并行更新模型中，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。具体来说，每次选择一个图块来加载时，分配一个概率p来选择包含进度最慢的行走的图块，即具有最小的行走步数，并且以概率$1-p$加载具有最多行走的图块。

3.3 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，将计算从行走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“像行走者一样思考”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。

3.3.1 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。

3.3.2　步进交错技术

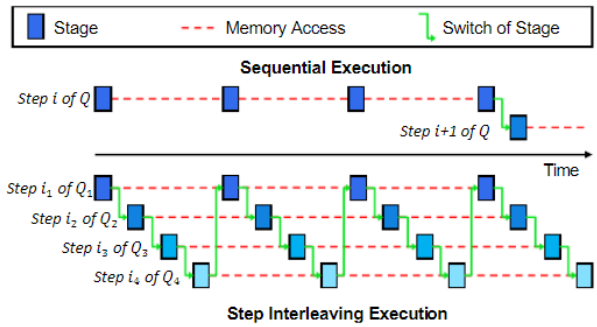

图6　顺序运行与步进交错运行

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6 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下面的公式来表示，其中$W_{hide} = max(W_L - kW_S - (k-1)W_C, 0)$。

$$W_{gain} = mW_L - (m+\overline{m})W_S - mW_{hide}$$

从该公式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}。

3.3.3　状态切换机制

ThunderSW 用阶段依赖图(SDG) 来对步骤中一系列操作的阶段进行建模。如图7 所示，SDG 中的每个节点都是一个包含一组操作的阶段，边表示它们之间的依赖关系。给定操作顺序，分两步构建SDG，抽象阶段（节点）和提取依赖关系（边）。

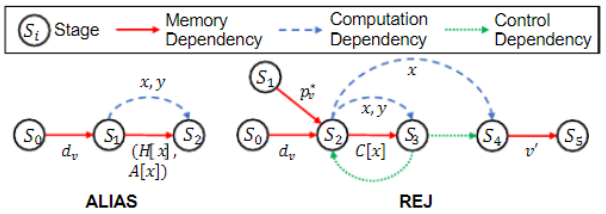

图7　状态依赖图

定义阶段：由于ThunderSW 通过切换查询的执行来隐藏内存访问延迟，阶段的约束是每个阶段最多包含一个内存访问操作，而消耗数据的操作在后续阶段。为了便于切换的实现，将包含跳转操作的操作视为单个阶段。

下面介绍SDG 下步进交织的实现。此时需要一个高效的开关机制。例如，禁止使用多线程，因为线程间上下文切换的开销以微秒为单位，而主存延迟以纳秒为单位。由于每个线程往往会进行许多RW 查询，因此我们在单个线程中的各个阶段之间切换执行。

根据它们是否属于SDG 中的周期将SDG 的阶段分为两类。对于不在循环中的阶段（称为非循环阶段），查询只访问它们一次以完成移动。给定一组查询Q，以耦合方式执行它们。特别地，一旦一个查询$Q_i \in Q$完成一个阶段S，就会切换到下一个查询$Q_{i+1} \in Q$来处理S。在所有查询完成S后，进入下一阶段。相反，周期中的阶段（称为周期阶段) 可以访问不同查询的不同时间。为了处理不规则性，以解耦的方式处理它们。具体来说，每个查询Q都记录了要执行的阶段S。当切换到Q，执行S，设定下一阶段的S完成后，基于SDG，并切换到下一个查询Q。因此，每个查询都是异步执行的。

针对查询中不同阶段之间的数据通信，基于SDG 创建了两种环形缓冲区，其中计算依赖边指示需要存储的信息。特别地，任务环用于跨查询所有阶段的数据通信，而搜索环用于处理循环阶段。由于需要显式地记录周期阶段的状态并控制它们的切换，处理周期阶段不仅会导致实现的复杂性，而且会产生更多的开销。NAIVE 和ALIAS

的SDG 没有循环阶段，因为它们的生成阶段没有for 循环，而ITS 、REJ 和O-REJ 的SDG有。

3.4 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采样和随机游走算法, 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。

3.4.1 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)。

3.4.2 GPU 采样优化

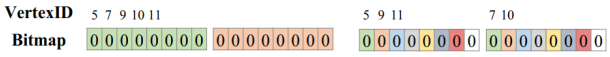

图8 连续位图和跨步位图

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8 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。

3.4.3 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 将执行相同的任务，并且不需要GPU 之间的通信。

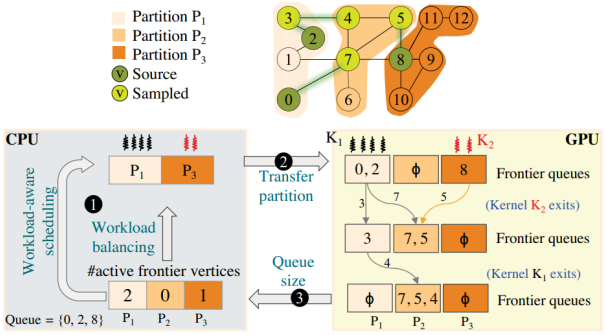

图9 工作负载感知的图分区调度

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9 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9 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9 中黑点3 所示。

3.5 FlashMob

目前最先进的随机游走系统设计方法集中在算法的改进上，例如前面所介绍的改进边采样

的KnightKing，支持大图核外游走的GraphWalker，它大大降低了复杂随机游走算法的内存需求，使非常大的图能够在内存中处理。然而，一旦主要的游走任务(即对下一个节点进行采样)能够在内存中处理，所有现有的系统都要付出随机访问的时间代价。除了随机访问，目前的系统还会随意地执行指针追逐。另外，现有系统无一例外地逐个处理行走者: 在迭代过程中，所有(活动的)行走者轮流采样，并从其当前节点的邻接表中选择一条边。这样的做法虽然直观，但会造成巨大的资源浪费。与图处理任务不同，图随机游走通过从潜在的许多候选边中仅选择一条边执行操作，因此它的处理更稀疏。因此，来自采样缓存行的大部分内容(通常存储十几条或更多条边)会被丢弃。这导致高速私有L2缓存中数据的低利用率，共享L3缓存中数据的低重用率，并增加了DRAM的总流量。FlashMob 系统的设计旨在减轻这些问题。

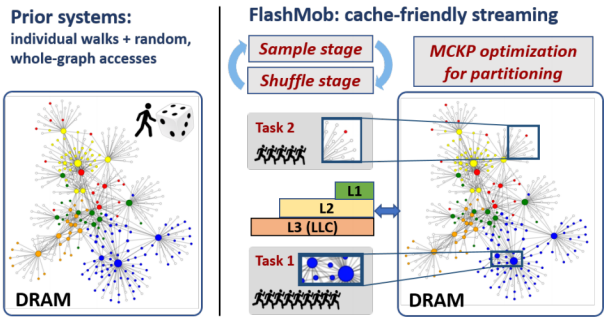

图10　FlashMob与现有系统的设计对比

由于现实世界的许多图具有幂律性，少数高度数顶点会吸引大量行走者。与统一处理所有节点的现有系统不同，FlashMob 的设计是基于这种高度倾斜的流量模式，图10 描述了该系统的架构，以下介绍该系统的工作流程。

与现有的随机游走实现不同，FlashMob 不会跟随行走者访问整个图，即使有足够的内存空间。相反，它将所有顶点按度降序排序，并将它们切割成许多顶点分区。图10 显示了两个这样的分区，一个包含几个高度顶点，另一个包含很多低度顶点。一个线程一次只处理一个任务，将当前在一个分区上的所有行走者移动一步。这显然需要将行走解耦为阶段: 采样和洗牌。另外，所有现有的解决方案都是单独移动行走者，FlashMob 利用高度顶点处的大流量，按批处理位于相同位置的行走者。它对边进行预采样，而不是为每个行走者单独获取下一条边，从而实现更便宜的采样生成，并充分利用存入预采样边的缓存行。

FlashMob 将顶点划分与行走者批处理相结合，极大地提高了访存效率: (1)使各任务的工作集与缓存大小相匹配; (2)提高顺序访存的数量。本质上，它在CPU 缓存内执行“核外”处理，并使用DRAM 进行快速数据流。

3.5.1　频率感知顶点分组与边采样阶段

FlashMob 的一个关键设计变化是对图进行划分，并批处理位于每个分区内的行走者。考虑到随机游走中顶点的度与它们被访问的频率总体上高度相关，FlashMob 按照顶点度的降序排列输入图中的顶点。

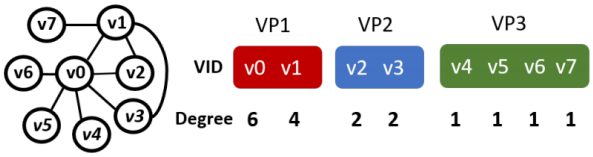

图11　顶点分区示例

可变大小的顶点划分。排序后的顶点数组被分割成大小不等的连续顶点分区(简称VPs)。图11展示了一个样本图，它的8 个顶点按度排序并切割成3 个VPs。VPs 构成基本任务单元，其大小根据图特征和系统资源约束进行优化。

FlashMob 的随机游走迭代的边采样阶段执行的核心任务是: 为每个行走者找到下一条移动的边。具体来说，通过将具有相似度的顶点分组，FlashMob 用不同的策略来处理不同的分组，对于一个给定的VP，FlashMob 选择一种采样策略，即预采样(PS)或直接采样(DS)，从而采用不同的数据组织和访问模式。

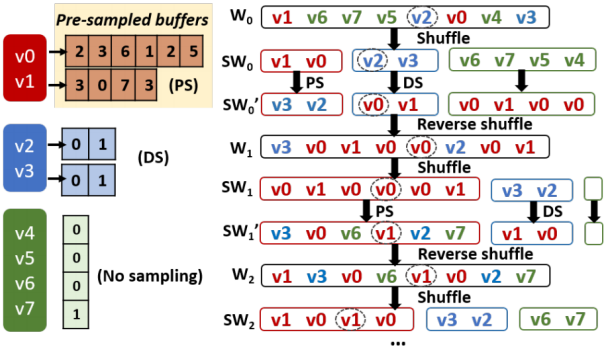

图12　行走者移动过程

对于这两种策略，第i轮迭代中，FlashMob 线程每次分配一个VP 和一个排序的行走者数组SW_i的连续块，该连续块存储了当前该分区内的所有行走者。线程扫描这个SW块，按顺序更新所有的行走者。一旦一个行走者从当前位置确定了它的下一条移动的边（无论是通过PS还是DS），FlashMob 都会用它下一步的顶点ID 覆盖

其存在SW_i中的顶点ID。在一个迭代的末尾反向洗牌之后，SW_i成为W_{i+1}，准备作为下一个洗牌阶段的输入。在图12 中，W1 和W2 分别在执行第一步和第二步后，以原始行走者的顺序存储所有行走者的位置。

预采样(PS): 该策略针对访问频率较高的节点设计，通过批处理同一节点上的行走者来最大化缓存利用率。其主要思想是提前对许多边进行采样，这些边被许多位于同一位置的行走者依次消耗。对于采用PS 策略的VP，每个顶点分配一个预采样的边缓冲区，行走者从中检索采样的边。如前所述，顶点被访问的频率与其度强相关，因此设置顶点v的预采样缓冲区的大小为$d(v)$. 在给定的VP中，处理任务的线程负责在预采样的边缓冲区为空时重新填满它们。在图12 中，只有第一个VP 采用PS，其中0 和1 的边缓冲区大小分别为6 和4。

PS 解耦了边样本的生产和消费阶段，虽然这会产生额外一轮边样本存储/检索操作，但通过批处理操作，整体内存访问的效率显著提高。

在生产阶段中，FlashMob 连续掷骰子，以边转移概率重新填充其预采样的边缓冲区。此操作一次只对一个顶点做，在此期间，该顶点的边(每条边平均采样一次)可以驻留在缓存中，以允许快速随机读取和向缓冲区中的顺序写入流。

在消费阶段中，被填充的缓冲区将里面的采样边给予$d(v)$个从顶点v离开的行走者。例如，在图12 中，0上的行走者会按顺序跟随采样边到达顶点$v2, v3, v6$等。

直接采样(DS): 直观地说，预采样的好处随着顶点度的减少而减少。在极端情况下，对于只有一条边的顶点，如图12 中的VP3，根本不需要边采样。对于度为2 的顶点，如图12 中的VP2，直接用它们的两条边存储和采样可以节省空间和时间。因此，FlashMob 提供了DS 选项，即行走者当场掷骰子，从邻接表(通常较短)中选择出边。在图12 中，无论何时一个行走者位于VP2 内时，它随机从一个顶点的两条出边采样。例如，第5 个行走者从其初始位置$v2$选择的$v0$作为下一个顶点。

除了紧凑的边存储，DS 允许FlashMob 利用其他优化。真实世界的图的长尾会产生许多度相等的顶点划分(比如上面的VP2 和VP3)。高度分区必须采用传统的CSR 图表示，其中采样顶点v的邻接表需要一次随机访问来读取顶点v的度，低度分区允许更简单的索引方式，跟踪分区的一个或几个不同的度值，使它能够使用简单的计算快速访问和采样。

3.5.2 基于动态规划的优化

FlashMob 提出了一种方法来划分顶点和分配采样策略，将其描述为一个组合优化问题，并获得了一个高效的动态规划解。

问题的定义。一个输入图具有按度降序排列的顶点，顶点ID 从0 到$(|V|-1)$，ID 为0 的是度数最高的顶点。为了缓存友好的边采样，FlashMob需要将顶点切割成连续的VPs，可以是不同大小的。主要的权衡在VP 的大小和数量之间: 较小的VP 适合更快的缓存，但好处很大程度上取决于VP 内的平均度数和行走者密度等因素; 更多的VPs 会增加洗牌开销，要么是由于不适合缓存，要么是由于增加了洗牌的级数。

问题简化和映射。由于FlashMob 的设计目标是适应不同大小的图，因此提出了两种近似方法来简化上述问题并减少解搜索空间。首先，将已排序的顶点列表分为G组, G是一个超参数(设置在64到128 之间)。除了最后一组，所有的组都是大小相等的，为便于索引，都包含2 的幂次的顶点数量。因为顶点是按度排序，组内的平均度与个体成员的度不会有太大的差异。一个组的成员顶点将再次被切割成2 的幂次大小的等大VPs，而顶点分区的大小在组间可以不同。其次，决定采样策略: 每个VP的顶点被分配使用PS 或DS。

然后该问题就可以归结为经典的组合优化问题，称为多选择背包问题(MCKP)。MCKP问题，即将一组物品(每个物品都有利润和权重)划分为不同类别。目标是从每一类物品中选择一件物品，在满足总重量限制的同时，使总利润最大化。

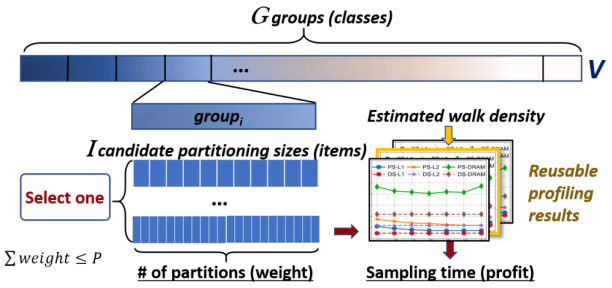

图13 FlashMob 分组问题映射为MCKP 问题

图13 展示了分组问题如何映射为MCKP 问题。G个组映射为G个类，对于每个类，候选物品是VP 大小和每个VP 的采样策略的二维组合。例如，一个物品“(PS，DS，PS，DS)”指定该组有4 个大小相同的分区，每个分区都有给定的采样策略。由于VP 大小要求是2 的幂，因此在一个组内，候选大小相当有限。对于每个候选VP 大小，将其利润定义为其总采样成本的负值，即一组内所有VP的采样时间之和，每个VP 的采样时间是其PS 和DS

时间之间的较低值。另一方面，其权重是使用这个VP 大小对应的组内VP 的数量。FlashMob 的洗牌阶段作为超参数P发挥作用，给出了总重量限制。在一台给定的机器上，FlashMob 将P设置为允许将洗牌任务放入L2 缓存中的vp 总数。

4 总结

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也逐渐提出，本文介绍了其中五个系统。

KnightKing 是第一个通用的分布式图随机游走引擎。它提供了一个直观的以行走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游走算法的代价。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。

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。

ThunderRW 是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随机游走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%。

C-SAW 是一种新颖的、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。

FlashMob 揭穿了长期以来的假设，即大型图上的随机游走需要随机DRAM 访问来解决。通过仔细的分区、重排和对操作的批处理，从随机游走算法中挖掘出隐藏的空间和时间局部性。实验结果显示，现代服务器的缓存即使对于具有随机和不规则计算的数据驱动程序也能实现高效的“缓存外”处理，由于串行和随机DRAM 访问之间的巨大差异，执行更多的顺序扫描是值得的，这最终可以节省时间和实际的DRAM 流量。

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。

致 谢 感谢施展老师、童薇老师和胡燏翀老师三位老师的指导，感谢同组的李真理、燕冉、田志鹏三位同学的分享。

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]// USENIX Annual Technical Conference. 2020.

[3] Pandey S , Li L , Hoisie A , et al. C-SAW: A Framework for Graph Sampling and Random Walk on GPUs[J]. 2020.

[4] Sun S , Chen Y , Lu S , et al. ThunderRW: An In-Memory Graph Random Walk Engine (Complete Version)[J]. 2021.

[5] Ke Yang , Xiaosong Ma , Saravanan Thirumuruganathan , et al. Random Walks on Huge Graphs at Cache Efficiency. SOSP 2021: 311-326.