分　数：	
评卷人：	

華中科技大學

研究生（数据中心技术）课程论文（报告）

题 目：图划分-混合边划分系统综述

学　　号 Y202102012

姓　　名 郑一帆

专　　业 计算机科学与技术

课程指导教师 施展 童薇

院（系、所） 研究生院

2022 年 1 月 7 日

图划分：混合边划分系统综述

郑一帆[1)]

[1)](武汉数字工程研究所-709 所，武汉 430074)

摘　要　当前，图结构化的数据无处不在。为了发掘图中蕴含的信息，需要对这些图数据经行处理计算，而图划分则是图处理工作的借助于分布式系统前置工作。在分布式系统内部对大型图结构化数据进行管理和处理，以较小的通信开销和查询用时解决了图划分问题。但最理想的图划分是 NP-hard 问题，不考虑计算时间复杂度，另一个重要的考虑因素是内存开销。现实世界中的图通常具有巨大的尺寸，因此将完整的图片加载到内存中进行划分即不经济也不可行。目前，主要依赖流式划分算法来减少内存开销。虽然最先进的流式划分算法在一些图上实现了较高的划分质量，但它们仍然不能完全与内存中的划分算法竞争。综上，研究人员提出了一种新的系统，混合边划分（HEP），它可以按照适应部分内存的方式对图进行划分，同时产生较高的划分质量。HEP 可以通过将图的边集分成两个子集来灵活地调整内存开销。其中一个子图用一种新颖、高效的 NE++内存中划分算法进行划分，而另一个子图则用流式划分方法进行划分。通过对现实世界中的大型真实图片进行评估，结果显示，在许多情况下，HEP 同时优于内存中划分和流式划分。因此，相较于现有诸多无法微调内存开销解决方案，HEP 是一个极具吸引力的替代方案。

关键词　图划分; 边划分; 内存约束; 大规模图处理

Graph partitioning: Hybrid edge partitioning system summary

Zheng Yifan[1)]

[1)](709th Research Institute of China State Shipbuilding Corporation , Wuhan 430074)

Abstract　Currently, graph-structured data is everywhere. In order to discover the information contained in the graph, these graph data need to be processed and calculated, and graph division is the pre-work of the graph processing work with the aid of the distributed system. The large-scale graph structured data is managed and processed in the distributed system, and the graph division problem is solved with less communication overhead and query time. But the most ideal graph partition is the NP-hard problem, without considering the computational time complexity, another important consideration is the memory overhead. Pictures in the real world usually have a huge size, so loading a complete picture into memory for division is neither economical nor feasible. At present, it mainly relies on streaming partitioning algorithms to reduce memory overhead. Although the most advanced streaming partitioning algorithms achieve high partition quality on some graphs, they still cannot fully compete with partitioning algorithms in memory. In summary, the researchers proposed a new system, Hybrid Edge Partition (HEP), which can divide the graph in a way that adapts to part of the memory, and at the same time produce a higher quality of division. HEP can flexibly adjust the memory overhead by dividing the edge set of the graph into two subsets. One of the subgraphs is divided by a novel and efficient NE++ in-memory partitioning algorithm, and the other subgraph is divided by the streaming method. Through the evaluation of large real pictures in the real world, the results show that in many cases, HEP is superior to both in-memory partitioning and streaming partitioning. Therefore, compared to many existing solutions that cannot fine-tune the memory overhead, HEP is an attractive alternative.

Key words　Graph partitioning; Edge partitioning; Memory constrints; Large graph processing

1 引言

在过去十年中，出现了许多专门用于管理和处理图结构化数据的框架，如 Pregel、Giraph 、 GraphLab 、 PowerGraph 、 Spark/GraphX、Neo4j 和 Trinity。所有这些系统都有一个基本特性：它们将一个大型图分布在多台机器上，并计算可能跨越整个图的全局查询，因此为了回答这些查询，多台机器之间的通信是不可避免的。同时，查询过程中带来的通信量取决于图在这些机器上的放置方式：图的划分尺寸越小，通信量就越小，处理查询就越快。这就导致了了数学和计算机科学中一个经典问题的复兴，现在专门优化的图数据处理系统的背景下进行研究：图划分（graph partitioning）。

在上述诸多发展的推动下，图划分研究的重点已转向高度倾斜的图，即顶点的度分布大致遵循幂律分布的图。在许多现实世界场景中这种图片大量存在，如网络图和在线社交网络。虽然图划分问题的经典方式是围绕顶点划分的，即通过切割图的边来分离顶点。但已经证明，边划分在减少偏斜幂律图中的分布式查询处理通信量方面更为有效。在边划分中，通过切割图的顶点来分离边。不巧的是，边划分是 NP-Hard 问题。因此，研究人员已经提出了许多启发式算法来解决大型图的边划分问题。

现有的划分算法可以分为两类：内存中的划分算法和流式划分算法。内存中的划分算法将完整的图加载到内存中，因此，具有在任何时间将任何边分配给任何分区的充分灵活性；流式划分算法通过边流，一次只查看一个（或少量）边。这两种图划分方法没有那种是完全令人满意的。内存中划分算法即使在具有挑战性的图上也可能产生非常高的分区质量，但不足是会消耗大量内存。流式划分算法消耗的内存很少，但即使通过基于窗口的流和多通道流等复杂技术对其进行改进，它也不会在所有图形上产生与内存中划分算法相同的最佳分区质量。在当前的图划分系统中，用户必须从这两个选项中的一个，要么提供一台非常大的机器（或一组机器）来获得良好的分区质量，要么使用小型机器获得一般的分区质量。

该领域现有工作的另一个缺点是，许多图划分算法都是从一个非常偏理论角度进行描述的，并没有讨论它们具体设计和实现。这使得没有机会对其进行探索优化。例如，在 NE 算法的实施过程中，作者以一种未经优化相当暴力的方式解决了防止边“双重分配”的复杂问题（参见 NE 算法概述）。HEP 的作者提出了一种新的边划分混合方法，它可以微调内存消耗和分区质量之间的平衡。在这样做的过程中，挑战了划分算法的常见模式，打破了纯内存中划分和纯流式划分算法的二分法。相反，HEP 在不同的子图上执行不同的划分策略。使用一种称为 NE++的新型高效内存中划分算法对至少有一个低度顶点的边进行划分。但是，关联两个高度顶点之间的边使用状态流式划分。这样，在流阶段利用内存中划分的划分状态来提高整体的划分质量。

HEP 的贡献如下：

1）提出了混合边划分（HEP）：这是一种遵循上述混合划分模型轮廓的新型图划分系统。在许多情况下，HEP 同时优于内存中划分和流式划分。

2）提出了 NE++算法：这是对著名的 NE 算法一种新的、高内存效率和速度的扩展。特别地是，与 NE 相比，相同的划分质量，NE++的运行时间和内存开销明显更低。通过引入两种新的优化来实现这一点，即图剪枝和延迟边去除，这使 HEP 的内存中划分阶段变得非常节省资源。

3）详细描述了在内存中与流两个阶段图划分的设计和实现。这填补了文献中的一个空白，大多数的文献主要关注划分算法设计，很少讨论它们的有效实现。

4）通过对多达 640 亿条边的图处理结果的广泛评估，显示出 HEP 的高效性。此外，在许多场景下，使用 HEP 替代以前提出的划分算法，在最先进的生产级分布式图形处理系统 Spark/GraphX 上获得提速。

2 原理与优势

2.1 边划分

HEP 解决的问题设定与划分、目标图的类型以及资源效率有关[1,3,4]。

划分类型:假设有一个无向无权图$G = (V, E)$有一组顶点V 还有一组边$E \subseteq V \times V$。边划分的目标是把边 E 分为 k（$k > 1$， $k \in$ N）个部分，分别被称为分区 P = {P_1，P_2,…P_k} 的并集就是边集 E，当 $i \neq j$时，P_i 与 P_j 交集为空。分区的大小受到平衡约束。图 1 展示了$k = 2$ 个分区时在示例星形图上的边划分。当边(u,v)被分配给一个分区 P_i。u,v 都被 P_i 覆盖。对于分区 P_i，定义$V(P_i) = \{x|(x,y) \in P_i\}$ 为被 P_i 覆盖的顶点集。当一个顶点v被一个分区覆盖，我们也说这个这个顶点v在该分区中被复制。在图一的例子中，顶点 0 在两个分区上被复制，而其他顶点仅在单个分区上被复制。

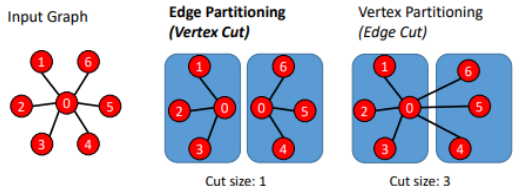

图 1：Edge partitioning VS Vertex partitioning

现在，图划分的优化目标是最小化复制因子 RF(P_1...P_k)。在给定示例下，复制顶点 0 会诱发两台机器之间的通信，这两台机器执行跨整个图的查询。例如，顶点 0 的状态需要在分布式图形处理算法的每次迭代中同步。通过降低复制因子，处理查询的分布式计算节点之间的同步量因此也降低了。

边划分与顶点划分截然不同，点划分中通过切割边来把顶点分配到相应分区（参见图 1 中的右侧）。这里，边切割带来查询处理中的通信量；机器借助于被切割的边交换顶点状态。从星形图的示例中可以看出，这样的边切割可能比相应的最佳顶点切割大出许多。已有研究人员分析了两种切割类型之间的差异，证明了大型幂律图上的顶点切割小于边切割。

图类型：在许多自然的图中，例如社交网络和网络图，顶点的度分布遵循幂律分布。在幂律图中，大部分边仅与少数具有非常高的度的顶点关联。可以利用此属性将边指定给分区，以便相对于度较低的顶点优先复制度较高的顶点。其基本原理是，度较高的顶点与更多的边关联，因此它们被复制的概率更大；通过关注诸多的低度顶点并给其一个较低复制因子，达到降低总体复制因子的目的。

效率：资源效率是我们需要的，例如，为了降低金钱成本。给定内存容量受限（可能小于图数据的大小）的计算节点，图形划分算法应该能够在尽可能低的运行时间内提供尽可能佳的复制因子。

2.2 混合边划分

HEP 是一种混合边划分系统，它将内存中划分与流式划分结合在一起。内存中划分可以实现较低的复制因子，因为相对于流式划分内存中划分受到的约束更小。但是呢，流式划分的内存开销较小，因为不需要把完整的图导入内存中。HEP 的目标是通过将目标图划分为两部分，一部分采用内存中划分算法，另一部分采用流式划分算法。为了实现该目标，必须找到一个合适的子图，可以在该子图上进行流式划分，同时也不会产生太高的复制因子。

3 研究进展

3.1 HEP 概览

通过考虑顶点的度，将图分为两个子图。为了验证这个想法，我们在两张不同的现实世界图上进行了实验（参见图 2）。我们同时使用流式划分器 HDRF 和内存中划分器 NE 进行 k = 32 的划分，测量具有特定范围（[1,10]、[11,100]等）度数的顶点的复制因子。在图 2 中，还绘制了每个图的度分布。我们可以观察到，由任一划分器划分获得的复制因子在很大程度上取决于顶点的度：低度顶点的复制次数远低于高度顶点。但是呢，一幅图中低度顶点的数量远高于高度顶点的数量。HDRF 和 NE 都是度感知划分器：

它们侧重于尝试减少多数低度顶点的复制因子，而不是少数高度顶点的复制因子[1,3]。

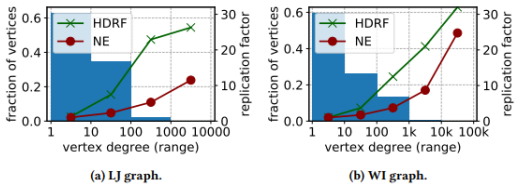

图 2：Degree VS replication factor for LJ and WI graph at k = 32 partitions

从某种意义上来说，HEP 采用的策略与现有的划分器相似，即试图为低度顶点实现较高的划分质量，以保持总体复制因子较低。但是，我们通过允许对高度顶点复制因子的妥协从这个新的方面提高了内存效率，以此来减少内存开销：

1）至少与一个低度顶点关联的边使用内存中划分算法进行划分。这显著降低了总体复制因子，因为在幂律图中，低度顶点的数量远高于高度顶点的数量。

2）与两个高度顶点关联的边(E_{h2h})使用流式划分算法进行划分。由于幂律图的偏斜性，即使高度顶点之间有许多边，受影响的顶点数量仍然很低。因此，可以使用流式划分在只影响少量顶点的情况下，对大量边进行划分。

但多高的度才算高度呢？太具有主观性，因此需要给“高”度一个。我们定义了一个阈值因子τ（tau）来分离低度顶点V_l 和高度顶点 V_h 。有两个约束条件 ：

$V_h \cup V_l = V$　　$V_h \cap V_l = \varnothing$

详情如下:

$$v \in \begin{cases} V_h & \text{if } d(v) > \tau * \varnothing_d, \\ V_l & \text{else,} \end{cases}$$

其中 $\varnothing_d$ 是图中所有顶点的平均度数，通过设置（threshold factor）τ，来判别高度与低度顶点，进而可以控制划分算法的内存开销：较低的τ意味着更多的顶点为高度顶点，因此关联高度顶点的边也就更多，也就意味着更多的边采用流式划分进行划分，达到减少内存开销的目的。

3.2 NE++设计

Algorithm 1 NE: Basic Algorithm.

```
1:  procedure INITIALIZE(p_i)
2:      v ← any vertex in V, such that v ∉ C
3:      MOVETOCORE(v)
4:  procedure PARTITIONGRAPH
5:      for each p_i ∈ P do
6:          while |p_i| < |E|/k do
7:              if S_i \ C ≠ ∅ then
8:                  v_min ← arg min_{v∈S_i\C} d_ext(v, S_i)
9:                  MOVETOCORE(v_min)
10:             else
11:                 INITIALIZE(p_i)
12: procedure MOVETOCORE(Vertex v)
13:     C ← C ∪ v
14:     for each u ∈ N(v) \ (C ∪ S_i) do      ▷ N(v) is the set of v's neighbors.
15:         MOVETOSECONDARY(u)
16: procedure MOVETOSECONDARY(Vertex v)
17:     S_i ← S_i ∪ v
18:     d_ext(v, S_i) ← d(v)
19:     for each u ∈ N(v) ∩ (C ∪ S_i) do
20:         d_ext(u, S_i) ← d_ext(u, S_i) − 1
21:         d_ext(v, S_i) ← d_ext(v, S_i) − 1
22:         if |p_i| < |E|/k then
23:             p_i ← p_i ∪ (u, v)            ▷ assign edge to p_i
24:             REMOVEEDGE((u, v))
25:         else                              ▷ spill over to next partition
26:             p_{i+1} ← p_{i+1} ∪ (u, v)
27:             REMOVEEDGE((u, v))
28:             S_{i+1} ← S_{i+1} ∪ {u, v}
```

图 3：NE：Basic Algorithm

HEP 的第一个阶段是通过 NE++进行内存中划分。NE++基于著名的 NE（neighborhood expansion）算法，并对其进行了一种新的、高内存效率与运行速率的扩展。特别地是，与 NE 相比，NE++引入了两个新特性。

1）NE++采用修剪图表示，显著降低了内存开销。

2）NE++引入了惰性(延迟)边删除，这是一种无需急于记录过去已分配的边、高效的避免将同一边分配给多个分区的技术。

基本 NE 算法。在详细介绍对于 NE++技术改进之前，我们先快速回顾一下基本的 NE 算法见图 3 并分析其缺点[3,4]。

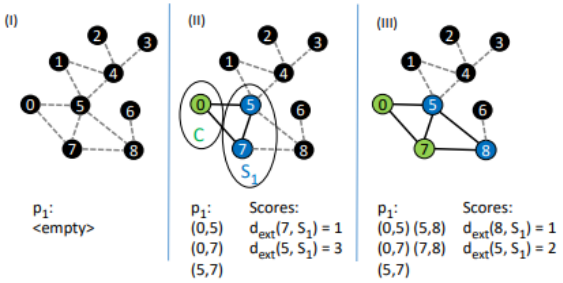

图 4：Example of expansion step

在图 4 中，我们提供了分区中第一个扩展步骤的详细示例。 C中顶点以绿色标记，S_1 中顶点以蓝色标记。我们进一步列出边集中已指定给 P_1 的和 S_1 中所有顶点的外部度数。

NE 的局限性。我们确定了 NE 的以下限制:

1）必须将完整的图形加载到内存中。这

可防止 NE 用于划分超出可用计算节点内存容量的图形。

2）NE 需要记录已将哪个边分配给哪个分区。这会导致大量内存、运行开销。

3.3 图剪枝

NE++采用了对压缩稀疏行（CSR）图表示的自适应。然而，当我们从输入图形文件（一个二进制边表）构建 CSR 时，我们省略了列数组（column array）的某些部分。特别的是，我们不在列数组中存储高度顶点的邻接列表。低阶顶点和高阶顶点之间的边仍然可以通过低阶顶点的邻接列表找到。但是，两个高次顶点之间的边根本不在列数组中表示。因此，在构建 CSR 时，我们将两个高次顶点之间的边写入外部文件。

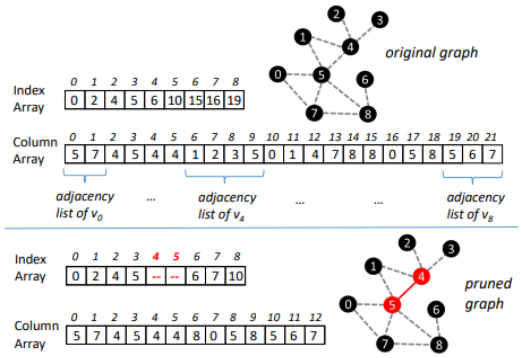

图 5：CSR representation of orginal and pruned graph

图 5 描述了一个示例。原始图形包含 9 个顶点和 11 个无向边；平均度为（空 d）为 2.4。在列数组中，这将导致 22 个条目。现在，度阈值为$\tau = 1.5$，度为 4 或更高的所有顶点都被视为高度顶点（在本例中，这些顶点为$v4$、$v5$)。在修剪图的列数组中$v4$ 和$v5$ 因此被省略了。为了不失去边($v4$ 、v5), 我们将其写入一个外部边文件，该文件随后通过流式划分算法进行划分。修剪后的图的列数组要小得多（在本例中，13 个条目而不是 22 个条目），也就是说，我们节省了内存。

为了使 NE 算法适应剪枝图表示，我们必须完全避免访问高度顶点的邻接列表。我们观察到，在 NE 中，高度顶点有保留在次集 S_i 中的趋势直到 P_i 分区扩展结束，因为与低度顶点相比，它们不太可能具有最低的外部度。在不同的现实世界图中使用 NE 划分且设置$k = 32$ 个分区下，我们对比了C中和 $S\backslash C$中顶点在正常化下的平均度。保留在次集中的顶点的平均度数非常高，而移动到核心集中的顶点的平均度数要低得多。换句话说，在整个分区扩展过程中，有许多高度点保留在次集S中，且不会移动到C。

在 NE++中，通过如下方式利用这个特性：我们将高度顶点预先分配给次集，并且不将它们移动到核心集（“不通过高次顶点进行扩展”），即高次顶点始终在次集中。这意味着我们无需迭代高度顶点的邻接列表，因此在 NE++中采用修剪图表示就足够了。与 NE 算法中将高度顶点移动到C 不同，NE++保持算法的原始行为[1,3]。

3.4 延迟边去除

有效边划分中的一个中心属性是，一条边必须被精确地分配一次，即精准地分配给一个单独的分区。但是在一个无向图中，一条边(u,v) 在 CSR 列数组中表示两次：如果v 是 u 的邻居，那么u 也是 v 的邻居。为了防止一条边被分配到多个分区，在进一步的分区过程中必须考虑那些已经分配了的边，即必须删除该边或将该边标记为无效。有两种非常简单直接的方法来实现这一点[1,3]。

方法一：可以从列数组中的两个不同位置删除该边，但是，这会导致计算开销，并带来较差的缓存局部性，因为列数组中的不同位置必须被顺序访问。

方法二：可以引入一个辅助数据结构用来保存每个边是否仍然有效的信息。NE 的参考实现遵循此方法。但是，这会带来额外内存开销，并且导致缓存局部性访问差，因为这样的数据结构不能在同一时间连续表示 u，v 两个顶点的邻接列表。

在 NE++中，我们引入了一种新的方法，称为延迟边去除，来更有效地解决此问题。延迟边去除既节省内存，因为它不需要引入辅助数据结构，也可以获得良好的缓存局部性，因为它只以连续方式访问邻接列表。在介绍此方法之前，我们首先分析在扩展算法的哪个阶段哪些顶点被访问。如果某个顶点

不再被任何其他分区再次访问，则删除该顶点邻接列表中的边就不是必须的。（因为都不会被访问了，所以删不删除也没有必要）

定理 1：在后续分区的扩展阶段，只有在分区扩展阶段结束时仍保留在分区次集中的顶点才能再次访问；移动到核心集中的顶点将不会再次访问。

由于移动到C 中的顶点的邻接列表将不再被访问，对于分区 P_i 那些保留在S_i 中的点。我们可以限制边去除。因此，我们引入了一个清理阶段，该阶段在每个分区完成后执行，它从列数组中删除这些可能在下一个分区 P_j, j>i 中再次访问的条目，但已经指定给的 P_i 的边。这仅影响次集 S_i 中顶点的邻接列表。

清理算法（算法 2）迭代 S_i 中所有顶点 v，并从列数组中删除那些已经分配给 P_i 的边（例如 v的邻居在 C 或者S_i 中). 通过将邻接列表的最后一个有效条目与待删除条目交换，然后减小邻接表的“大小”（一个指示有效条目数量的字段）。

可以高效地执行边去除，这是一个固定时间的操作。图 6 描述了执行清理算法前后的一个图形示例。该图显示了清理算法如何从图的其余部分“分离”核心集；后续扩展无法再进入图的核心集，因为清理算法会删除通向它的所有链接。因此，连接核心集中两个顶点的边都不会再次访问，因此，不可能对这些边进行双重赋值。正如 NE++中的启发式算法所尝试的那样保存S_i 尽可能小，在清理阶段只访问少量顶点。

Algorithm 2 Clean-up Algorithm.

```
1: procedure CleanUp(Partition p_i)
2:     for each v ∈ S_i do
3:         for each u ∈ N(v) ∩ (C ∪ S_i) do
4:             v.adjacency_list.Remove(u)
```

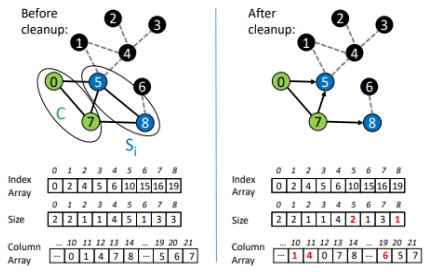

图 6：Clean-up phase Clean-up Algorithm and example

3.5 HDRF 改进

在 HEP 中，我们使用状态流式划分，并利用内存中划分的分区状态来提高分区质量。状态流式通过评分函数$\sigma: (e, p) \rightarrow s$对给定边 e 放置在不同分区 P_i 进行评分，然后，将边分配给产生最高分数 s 的分区。

在 HEP 实现中，我们使用 HDRF 评分函数，该函数考虑了顶点的度、每个分区的顶点复制情况以及每个分区的负载（以边的数量为单位）。每个分区的顶点的和顶点的复制情况作为 NE++的结果使用：一个顶点在分区 P_i 中被复制只有当它在S_i 时。这样，就克服了状态流式划分“不知情的分配问题”，状态流划分中的早期边被分配给一个分区，而不知道图的其余部分。HDRF 是最适合 HEP 流式阶段的分区算法，可以轻松集成，而无需任何调整。尽管如此，HEP 流式阶段也可以采用其它状态流边划分算法，如 Greedy 或 ADWISE。然而，贪婪策略的性能明显借助 HDRF 做得更好。ADWISE 投入额外的时间解决 NE++在内存阶段已经解决的不知情分配[1,4]。

3.6 复杂度分析

Name	Type	Time Complexity
HEP	Hybrid	$O(\|E\| * (\log \|V\| + k) + \|V\|)$
Greedy [28]	Stateful Streaming	$\Theta(\|E\| * k)$
HDRF [51]	Stateful Streaming	$\Theta(\|E\| * k)$
ADWISE [47]	Stateful Streaming	$\Theta(\|E\| * k)$
DBH [64]	Stateless Streaming	$\Theta(\|E\|)$
Grid [32]	Stateless Streaming	$\Theta(\|E\|)$
NE/NE++	In-Memory	$O(\|E\| * (\log \|V\| + k) + \|V\|)$
DNE [30]	In-Memory	$O(\frac{d*\|E\|*(k+d)}{n*k})$ with d = maximum vertex degree, n = number of CPU cores
METIS [34]	In-Memory	$O((\|V\| + \|E\|) * \log k)$

图 7：Time complexity of different partitioners

图构建阶段的时间复杂度为：O (|E| + |V |)
NE++阶段的时间复杂度为：O (|E| ∗ (log |V | + k) + |V |)
Streaming 阶段的时间复杂度为：O (|E| ∗ k)
空间复杂度（内存开销）为：V_i ∈V_l d (v_i) ∗ b_{id} + 6 ∗ |V | ∗ b_{id} + |V | ∗ （k+1）/8 bytes
与其它相关算法的时间复杂度对比见图 7。

3.7 大型图数据集

Name	$\|V\|$	$\|E\|$	Size	Type
com-livejournal (LJ)	4.0 M	35 M	265 MiB	Social
com-orkut (OK)	3.1 M	117 M	895 MiB	Social
brain (BR)	784 k	268 M	2 GiB	Biological
wiki-links (WI)	12 M	378 M	3 GiB	Web
it-2004 (IT)	41 M	1.2 B	9 GiB	Web
twitter-2010 (TW)	42 M	1.5 B	11 GiB	Social
com-friendster (FR)	66 M	1.8 B	14 GiB	Social
uk-2007-05 (UK)	106 M	3.7 B	28 GiB	Web
gsh-2015 (GSH)	988 M	33 B	248 GiB	Web
wdc-2014 (WDC)	1.7 B	64 B	478 GiB	Web

图 8：Real-world graphs

我们使用 7 种不同的图（参见图 8）具有不同大小和来源于不同的 web 存储库，由不同的组织进行了爬取。这些是社交网络图（OK，TW 和 FR）以及网络图（IT、UK、GSH 和 WDC）。我们选择这些图的原因如下：

第一：它们中的许多在相关文献中常用作基线。

第二：它们中的大多数都非常大（数十亿条边），因此即使在大型机器上，在内存开销方面也特别具有挑战性。事实上，我们表明并非所有的分区器都能在我们的评估平台上处理如此大小的图。

第三：这些图在划分的“容易程度”方面有很大的差异；对于其中一些，即使是最好的分区器也只能达到相对较高的复制因子[1,2]。

3.8 图划分结果

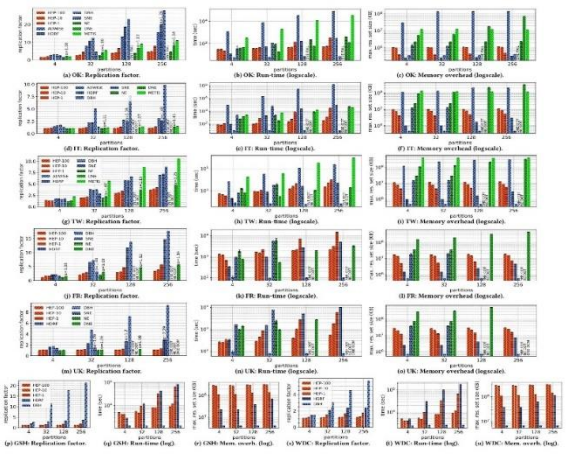

图 9：Perfermance results on real-world graphs

图9显示了所有的结果。观察结果主要如下：

1）HEP 的内存开销低于内存分区器 NE、DNE 和 METIS（见图 9（c、f、i、l、o））。不管τ的设置，内存分区器的内存开销高达一个数量级。同时，HEP 可以达到与 NE 竞争的复制因子（特别是在$\tau \geq 10$），它是在所有实验中具有最佳复制因子的分割器（参见图 9（a、d、g、i、m））。

与 NE 相比，提升的一个主要原因是延迟边缘删除方法，它允许我们删除 NE 中的一个辅助数据结构，即未排序的边缘列表，该列表跟踪每个边缘是否仍然有效。此外，HEP 的运行时间更好，因为我们使用了更高效的数据结构，即 CSR 表示。与 HEP 不同，NE 内部使用的未排序边缘列表在执行邻域扩展步骤时会导致大量随机访问和缓存未命中。

DNE 是可扩展边缘分区的最优技术，能够跨大型计算机集群进行扩展。然而，这是有代价的：DNE 需要比 HEP 多一个数量级的内存。此外，如果用于分区的资源有限（在本例中为 64 个硬件线程），HEP 的运行时可与 DNE 竞争；在某些情况下，甚至更低。此外，DNE 的分布式和并行性质导致复制因子的退化。在所有评估的环境中，HEP 比 DNE 获得更好的复制因子在所有的τ设定和在所有图上。最后，DNE 不能始终保持分区平衡，而 HEP 在所有情况下都实现了完美的平衡。

METIS 通常被认为是图形划分质量的“黄金标准”，通常用作基线。虽然 METIS 获得的复制因子在某些情况下（特别是在 IT 图上）相对较好，但 METIS 需要花费大量时间，因为它遵循具有粗化和细化阶段的多级分区方法。与 METIS 相比，HEP 速度更快，占用内存更少，并且实现了更好的复制因子。在某些情况下，HEP 在运行时的优势超过一个数量级，

e.g.在 TW 图上k = 32，HEP-100 为 15 分钟，METIS 为 10 小时（图 9（h）），而 HEP-100 的复制系数为 1.99，METIS 为 5.68（图 9（g））。

2）与流式分区器（ADWISE、HDRF、DBH 和 SNE）相比，HEP 实现了更好的复制因子。同时，可以减少 HEP 的内存开销，使其接近流式分区。通过设置较低的值τ，在

所有图和所有数量的分区上，我们观察到HEP 的内存开销显著减少。在$\tau = 1$、HEP 的内存

开销接近于流式分区（图 9（c、f、i、l、o、r、u））。这样，可以保持较小机器上的内存边界。然而，HEP 实现了比流式分区器更好的复制因子（图 10（a、d、g、j、m、p、s））。

3.9 相关算法总结

现有的图分区器不是内存分区器，就是流式分区器。内存分区器中最突出的类型是多级分区器，如 Chaco，Jostle，Scotch，Metis 或 Kahip。它们应用了三个如下步骤：首先，通过组合相邻顶点对图进行粗化。第二，对粗化图进行分区。第三，将粗化图的分区细化为原始图。各种多级分区器之间的主要区别在于，它们对每一步使用的算法。这些分类器关注分区质量，但是带来较高的开销。在我们的实验中，KaHiP 在大型图上耗尽了内存，而 METIS 在某些图上运行了数天。

Sheep 是一个分布式边缘分割器，用于构建和分割输入图的消除树。Spinner 和 XtraPuLP 是基于用于群落检测的标签传播算法的分布式和并行顶点分区器。DNE 分区器在复制因子、运行时、内存效率和可伸缩性方面优于这些分类器。HEP 在分区质量和内存开销方面优于 DNE。NE 是一种顺序分区器，与其他非多级分区器相比，它可以产生更高的分区质量。在 HEP 中，我们改进了 NE 的设计，并克服了内存开销和运行时间较大的缺点。

一些内存中的分割器，如 Sheep 和 Streaming NE（SNE），通过创建按顺序一次一个分区的输入图块来减少内存开销。但是，这会导致更长的运行时间，对于 SNE，也会导致分区质量更差。HEP 也做出了这样的权衡，但其性能优于以前的方法，因为它没有将相同的内存算法应用于输入图的后续块，而是将不同的算法应用于图的选定部分。这将最大化图中最关键部分的分区质量，即关联到低阶顶点的边，同时仅在少量高阶顶点上放松分区质量。

在上文中评估了大量的流式分区器。它们要么处理 hashing，要么在通过图形流时收集并利用分区状态。HEP 分区器在复制因子方面优于这些分类器。流模型的常见变体是重新流化，即通过图形流执行多个过程并细化每个过程中的分区，以及局部重新排序，即缓冲图形部分流并对其重新排序以提高分区质量。TLP 基于与 NE 类似的邻居扩展方案，但采用半流方式。作者声称空间复杂度较低，但 TLP 要求以不同的广度优先搜索顺序对图形进行多次排序和流式处理。这种开销对于实现具有竞争力的运行时来说似乎是不可接受的。

Fan 等人提出了一种方法，将给定的顶点或边划分细化为适合图形处理作业的混合划分。HEP 可以用作此操作的预分区算法。此外，Fan 等人提出了一种增量迭代顶点切割分割的方法，该方法也适用于 NE++。与图分区类似的一个问题是找到“顶点分隔符”，即从图中移除顶点会将其拆分为两个或多个断开连接的组件。

压缩技术，如 SlashBurn 或 Frame of Renference，可以减少图形处理算法的内存占用。然而，压缩的效果取决于图形数据。与此不同，HEP 可以通过τ调优以可预测的方式减少内存开销. 参考框架还受益于已排序的邻接列表，这与 NE++更改顺序的修剪机制不一致。

PowerLyra 是一个遵循混合分区和处理策略的图形处理系统：它将顶点分区应用于低度顶点，将边分区应用于与高度顶点相关的边。虽然这个想法与我们的想法相关，因为它的本质是以不同的方式处理低阶和高阶顶点，但 PowerLyra 完全在内存中。内核外图形处理系统，如 GraphChi、X-Stream、Chaos、GridGraph 和 Mosaic 通常对图形数据进行预处理，以增加访问局部性。这些预处理问题不同于我们针对 HEP 的分区问题。在内核外图形处理框架中实现 NE 算法效率低下，因为 NE 中的迭代次数与顶点数呈线性关系，而这些框架中的每次迭代都需要对整个图形进行完整的遍历[1]。

4 总结与展望

HEP 是一种新的用于边分割的混合系统。HEP 基于顶点的度将边集分离为两个子集，并使用一种新的内存算法 NE++和之后的告知状态流将它们分开。评估表明，与纯内存划分系统相比，HEP 可以更快地生成最优的复制因子，并且具有更少的内存开销，同时可以生成比流式划分更好的复制因子。最后，使用 HEP 进行大型图划分可以加快图形处理任务的运行时间。

在未来的工作中，研究人员的目标是通过关注并行性和分布来进一步提高 HEP 的性能。除此之外，探索 HEP 对超图（hypergraphs）的扩展。

HEP 通过在整合内存分区和流式分区，结合了这两个方面的优点。这为最终用户带来了以下好处：

1）灵活地将内存开销调整到适应手头的图形和现有硬件功能，而无需在分区器之间切换。而不是必须安装或甚至集成到另一个系统中许多不同的分区器，只需使用 HEP 和调谐旋钮（参数τ) 将图放入内存。

2）在内存分区因图太大而不可行的情况下，HEP 比流式分区产生更好的复制因子，尤其是在社交网络图上。

3）在内存分区理论上可行的情况下，使用 HEP，并将其设置为τ 产生良好的复制因子，并且比以前最先进的分区算法 NE 快得多。

4）HEP 也给我们算法设计等工作带来了思路，短时间内不可能做到某个领域颠覆性的创新与突破，但可以整合已有的诸多解决方案，他山之石，可以攻玉，取百家之长为我所用，亦是一种解决思路。

参考文献

[1] Ruben Mayer and Hans-Arno Jacobsen. 2021. Hybrid Edge Partitioner: Partitioning Large Power-Law Graphs under Memory Constraints. In Proceedings of the 2021 International Conference on Management of Data (SIGMOD ’21), June 20–25, 2021, Virtual Event, China. ACM, New York, NY, USA,14pages.

[2] Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. 2014. GraphX: Graph Processing in a Distributed Dataflow Framework. In 11th USENIX Symposium on Operating Systems Design and Implementation.

[3] Chenzi Zhang, Fan Wei, Qin Liu, Zhihao Gavin Tang, and Zhenguo Li. 2017. Graph Edge Partitioning via Neighborhood Heuristic. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD’17).

[4] Fabio Petroni, Leonardo Querzoni, Khuzaima Daudjee, Shahin Kamali, and Giorgio Iacoboni. 2015. HDRF: Stream-Based Partitioning for Power-Law Graphs. In Proceedings of the 24th ACM International Conference on Information and Knowledge Management (CIKM ’15).