用于层次聚类的并行过滤图

Shangdi Yu*MIT CSAIL*

Julian Shun
MIT CSAIL

arXiv:2303.05009v1 [cs.DC] 9 Mar 2023

摘要—给定一组对象两两之间的所有权重（距离），过滤图仅保留重要权重，从而提供一种稀疏表示。此类图可输入图聚类算法，以生成层次化簇。特别是，研究表明，在过滤图上运行有向气泡层次树（DBHT）算法，能够为时间序列数据生成优质的层次化簇。

我们提出了一种用于构建三角化最大过滤图（TMFG）的新并行算法，该算法可生成适用于 DBHT 的有效输入；此外，还提出了一种可扩展的 DBHT 并行生成算法，并针对 TMFG 输入进行了优化。除了将原始 TMFG 构建过程并行化（其并行性有限）之外，我们还设计了一种新算法，每轮插入多个顶点，以提高并行性。我们证明，与原始 TMFG 相比，新算法生成的图质量相近，但生成速度快得多。我们提出的 TMFG 和 DBHT 并行算法比当前最先进的实现快 136–2483 倍；在启用超线程的 48 个核心上，自相对加速比最高可达 41.56 倍，并且与标准的平均连接和完全连接层次聚类算法相比，能取得更好的聚类结果。我们还表明，在一个数据集上，我们的算法生成的簇与人类专家分类高度吻合。

I. 引言

聚类是一种无监督机器学习方法，已广泛应用于金融、生物学和计算机视觉等许多领域，用于发现数据集中的结构。人们通常希望探索不同分辨率下的簇。层次聚类算法可用于生成一棵树，也称为树状图，以表示不同尺度下的簇。

对一组 n 个点运行度量聚类算法，通常需要处理 $\Theta(n^2)$ 个两两距离，对于大型数据集而言，计算开销过高。一种提高效率的方法是使用过滤图，仅保留部分两两距离，然后将生成的图传递给图聚类算法。过滤后的图减少了需要考虑的距离数量，同时保留了局部和全局最重要的特征。简单地

在实践中，移除所有权重低于某个阈值的边可能表现不佳，因为该阈值可能需要进行大量调优，而且一条边的重要性不仅由其权重决定，还取决于其在图中的位置。已有研究提出了多种构造过滤图的方法，包括 k近邻图、 [1],最小生成树 [2], [3], 以及加权极大平面图 [4], [5]。

一种加权极大平面图的变体，称为平面最大过滤图（PMFG） [4], ，已被证明在实践中与有向

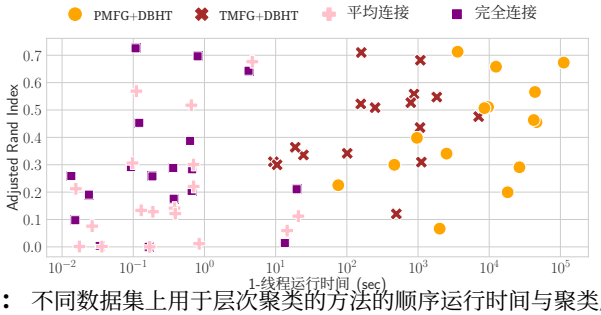


图 1： 不同数据集上用于层次聚类的方法的顺序运行时间与聚类质量。

气泡层次树（DBHT）技术结合使用时效果良好，尤其是在金融 [6]–[8] 和生物学 [9], [10] 数据集上。PMFG通过反复在权重最高的点对之间添加一条边，同时保持平面性来构建。DBHT技术根据输入图中的某些三角形以及各种最短路径计算来构建树状图。它的优点是无需关于数据的先验信息，也无需调整参数。然而，最先进的PMFG和DBHT算法都是串行的，无法扩展到大型数据集。Massara 等人 [5] 提出了三角剖分最大过滤图（TMFG），这是一种可更高效生成的极大加权平面图变体。与PMFG反复添加单条边不同，TMFG通过反复向一个三角形添加单个顶点及其与三角形三个顶点之间的三条边来构建，同时保持平面性。换句话说，我们要寻找一个尚未插入的顶点 x ，使其与图中已构成三角形的三个顶点“接近”，然后通过添加从 x 到这三个顶点的三条边将其插入 x 。然而，最先进的TMFG实现同样是串行的；据我们所知，尚未有人评估使用TMFG构建的DBHT的聚类质量。

PMFG、TMFG 和 DBHT 已被证明在一种重要的数据类型上表现良好，即时间序列数据。时间序列数据广泛应用于多个领域，包括金融、信号处理、生物学、天文学和天气预报。为了从数据中提取见解，人们通常会关注不同时间序列之间的相关性，并基于这些相关性对数据进行聚类。相关矩阵 存储了每对时间序列之间的相关性。构建相关矩阵的过滤图，对于实现高效且可扩展的聚类至关重要。图 1 展示了 PMFG 和 TMFG 与 DBHT 结合使用时的运行时间和聚类质量（使用调整兰德指数 [11]），并将其与标准的平均连接和完全连接层次聚类方法进行比较，针对的是一组时间序列数据集。

对于所有方法，树状图的切割方式都使得生成的簇数与真实聚类的数量相同。我们看到，尽管 PMFG 和 TMFG 的运行时间高于平均连接和完全连接方法，但它们能够生成质量更高的簇。因此，当人们愿意投入更多时间以获得更高质量的簇时，PMFG、TMFG 和 DBHT 更适合用于层次聚类。

为缩短 TMFG 和 DBHT 的运行时间，我们提出了新的并行算法和快速实现方案，用于构建 TMFG，并提出了一种针对 TMFG 优化、可扩展的并行 DBHT 算法。除了并行化原有的 TMFG 构建（其并行性有限）之外，我们还设计了一种新算法，每轮插入多个顶点，以实现更高的并行性。我们证明，新算法生成的过滤图质量与原始 TMFG 相近，但速度快得多。该算法的主要挑战在于解决向 TMFG 插入多个顶点时产生的冲突。在 TMFG 构建过程中，我们根据边权重，为每个顶点指定一个三角形，以解决顶点之间的冲突。我们的 DBHT 算法利用了 TMFG 的特殊拓扑结构。我们能够在构建 TMFG 的同时，借助一种特殊不变量动态构建一棵有根无向泡树，并利用该不变量通过递归过程高效地为泡树定向。这样便将这两个步骤的复杂度从二次复杂度（原始 DBHT 算法所需）降至线性复杂度。

我们将并行算法的速度和质量与并行化的平均连接和完全连接层次聚类算法进行比较。作为基准，我们还与 k-均值进行了比较；这是一种非层次聚类算法，只能在单一分辨率下生成簇。在由时间序列和图像数据生成的 16 个数据集上，我们发现，使用 TMFG 的 DBHT 生成的簇与使用 PMFG 的 DBHT 生成的簇相似，有时甚至更好。它生成的簇也优于平均连接聚类和完全连接聚类，并且在大多数数据集上与 k-均值具有竞争力。

我们证明，我们的新算法比基于 PMFG 的顺序 DBHT 最快可提高 15589 倍，并且其调整兰德指数得分最高可比凝聚聚类算法高 0.65。我们表明，在来自 2013–2019 年美国股市股价的时间序列数据集上，基于 TMFG 的 DBHT 能够生成与人类专家分类良好对齐的簇；在我们的实验中，它生成的簇优于原始 TMFG 算法。使用带超线程的 48 个核心时，我们构建 TMFG 变体和 DBHT 的新算法比最先进的 TMFG 实现快 136–2483 倍，并且在 48 个核心上实现了最高 41.56 倍的自相对加速比。

我们总结如下贡献：

- 我们设计了一种新的并行算法，用于构建 TMFG 的变体，从而生成高质量的图。
- 我们设计了一种新的并行算法，用于构建针对 TMFG 类输入进行优化的 DBHT。
- 我们通过实验表明，与最先进的算法相比，我们的并行算法可实现显著加速，同时生成质量相当或更优的簇。

我们的源代码和数据可在 https://github.com / yushangdi/par-filtered-graph-clustering 获取。

二、背景

我们简要介绍 PMFG 和 TMFG 的顺序构建过程，并说明所用的记号和基本操作。

平面图可以绘制在平面上，使各边互不相交。PMFG 和 TMFG 都是完全无向加权图的极大平面子图（“极大”是指在不违反平面性的前提下，无法向顶点集合中添加任何边）。它们是 NP难加权最大平面图问题的一种近似解；该问题要求保留边的权重之和最大化 [12]。完全加权图输入也可以表示为相似度矩阵 S，其中 $S[i, j]$是图中边 (i, j) 的权重。

平面最大过滤图（PMFG）。构建 PMFG 的顺序算法 [4] 首先按边权重对所有边进行排序。随后从空图 G 开始，按照排序顺序从权重最高到最低逐条考察边。只有在添加某条边不会破坏 G的平面性时，才将其加入图中；这通过运行平面性测试算法来检查。由于需要运行平面性测试算法 $\Theta(n^2)$ 次，这一过程的计算开销很大。

三角剖分最大过滤图（TMFG）。提出 TMFG [5] 是为了提高平面最大过滤图构建的效率。原始顺序 TMFG 算法不再逐条考察边，而是在每次迭代中，将一个顶点 x 加入正在构建的子图的顶点集合，同时添加三条边（每条边都从 x 连接到子图平面嵌入中某个三角面的三个顶点之一），并在后续迭代中不再考虑该顶点。选择能够使边权重总和增加最多的顶点—三角形组合。这样一来，TMFG 只需进行$\Theta(n)$ 次迭代，而 PMFG 则需要 $\Theta(n^2)$ 次。算法从加权度最大的四个顶点及连接它们的全部六条边开始。根据定义，该算法确保添加的边不会破坏平面性，因此无需进行平面性测试。图 2(a)展示了一个 TMFG 示例。如果还有其他顶点需要插入，它们可以加入任意一个面，而这些面都是三角形。

有向气泡层次树（DBHT）。在获得TMFG或PMFG后，我们可以使用DBHT [13]算法从中生成用于层次聚类的树状图；事实证明，该方法能够为金融时间序列 [6]–[8] 和生物 [9], [10]数据生成高质量的树状图。原始的顺序DBHT算法首先构建一个气泡树 [14],，这是一棵树，其中节点对应于平面子图，而两个节点之间的边对应于原图中将相应平面子图分隔开的三角形；

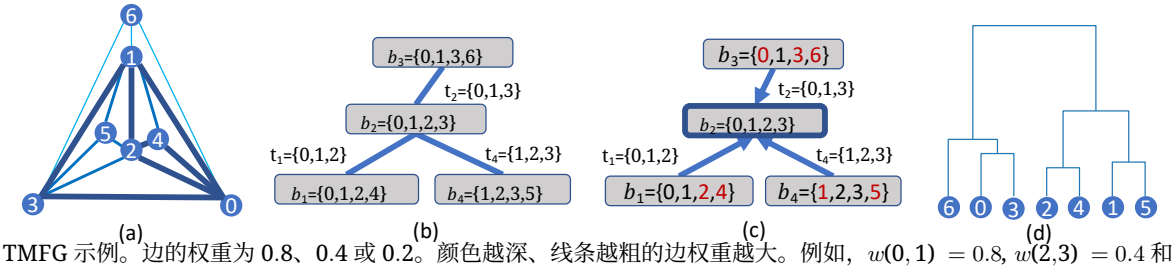


图 2： **(a)** TMFG 示例。边的权重为 0.8、0.4 或 0.2。颜色越深、线条越粗的边权重越大。例如，$w(0,1) = 0.8$, $w(2,3) = 0.4$ 和 $w(0,6) = 0.2$。**(b)** 无向气泡树。每个气泡节点都是一个灰色方框，节点之间以蓝色边连接。每个气泡都标有其中的顶点和一个名称 b_i。每条边都标有其对应的三角形和一个名称 t_i。**(c)** 有向气泡树。加粗的节点是此例中唯一的汇聚气泡。红色顶点被分配到标记它们的气泡中。**(d)** DBHT 的树状图。

DBHT算法随后通过计算分离面与其内部和外部之间的连接强度，为树中的边确定方向。接下来，该算法执行多次最短距离计算，将顶点分配给气泡，最后使用完全连接聚类生成层次结构。我们将在第五节中详细介绍DBHT算法，以及如何对其进行并行化和优化。

符号与术语。给定一个平面图 G 及其平面嵌入，G 的面是由边围成的极大区域，但外面除外，因为外面是无界的。例如，在图 2(a)中，$\{0,3,6\}$ 是该嵌入中的外面。有界面是内面。我们通过添加三条加权边，分别连接 v 与 v_x、v_y和 v_z，将顶点v 插入 到一个有三个顶点 v_x、v_y和 v_z 的三角面 t 中。将 v 插入 t 所产生的增益 是这三条边权重之和。设 V 为一个顶点集合。如果将顶点 v 插入面 t所产生的增益，在 V中的所有顶点里最大，则称顶点$v \in V$ 是面 t 对应的 V 中的最佳顶点。图 G 中边（i，j）的权重记为 w（i，j）。

如果移除一个面会使图不连通，则称该面为分离的 。气泡 是一个极大平面图，其中的3-团是非分离圈。如果我们为每个气泡创建一个气泡节点 ，并连接共享分离三角形的气泡节点，那么这些气泡节点及其连接边构成一棵树 [13], [14]，称为气泡树。我们用 C表示一个团，用 b表示气泡树中的一个气泡节点。

我们算法的输出是一棵称为树状图的树，其中每个节点的高度对应于其两个子节点所代表的两个簇之间的不相似度。树状图节点的高度 是由我们的算法计算得出的值，且节点的高度至多等于其父节点的高度。在不同高度处截断树状图，可得到对应不同尺度下簇的子树。

计算模型。 为了分析我们的并行算法，我们使用工作-跨度模型 [15], [16], ，这是一种用于分析共享内存算法的标准模型。算法的工作量 W 是操作总数，跨度 S是算法最长依赖路径的长度。顺序算法的工作量等于其运行时间。我们可以通过使用随机化工作窃取，将算法在 P个处理器上的运行时间限制为 $W/P+O(S)$

表 I： 并行原语。

运算符	描述	Work	Span
过滤	筛选出所需元素	$O(n)$	O(对数 n)
Sort	排序元素	$O(n)$	O(对数 n) 以高概率
最大值	查找最大值	$O(n)$	O(1) 以高概率
WRITEMIN	并发写入 最小值	$O(1)$	$O(1)$
WRITEMAX	并发写入 最大值	$O(1)$	$O(1)$
WRITEADD	原子性地添加	$O(1)$	$O(1)$

调度器 [17]。

并行原语 **[15], [18]**。表 I 总结了我们使用的并行原语。并行筛选接受一个数组 A 和一个谓词函数 f，并返回一个新数组，其中包含使 $f(a)$ 为真的 $a \in A$，其顺序与它们在 A 中出现的顺序相同。过滤的工作量为 $O(n)$，跨度为 $O(\log n)$。并行排序接受一个数组 A 和一个二元函数$f(a,b)$；当 $a <_k b$ 时，该函数返回 true，其中 $<_k$ 是一个全序。它返回一个新数组，其中包含按非递减顺序排列的 $a \in A$ 。排序的工作量为 $O(n\log n)$，跨度为 $O(\log n)$。对于范围为$[1,\ldots,O(n \log^c n)]$ [19]的整数，并行整数排序可在高概率下以 $O(n)$ 的工作量和 $O(\log n)$ 的跨度完成（w.h.p.）[1] 。并行最大值接受一个数组A，并返回其中的最大元素 A。高概率下，可用 $O(n)$ 的工作量和 $O(1)$ 的跨度计算最大值。WRITEMIN 是一种优先级并发写入操作，接受两个参数作为输入：第一个参数是要写入的位置，第二个参数是要写入的值；发生并发写入时，写入最小值 [20]。WRITEMAX 与之类似，但写入最大值。WRITEADD 是一种优先级并发写入操作，接受两个参数作为输入：第一个参数是要写入的位置，第二个参数是要以原子方式加到第一个位置处的值上的数值。我们假设这些优先级并发写入操作的工作量和跨度均为常数。

III. 相关工作

设计串行和并行层次聚类算法方面已有大量研究。最流行的层次聚类算法变体是层次凝聚聚类算法（HAC） [21]–[25]。HAC 的不同变体使用不同的连接函数来计算簇之间的距离。常见的连接函数包括完全连接，

[1]如果某个界以至少 $1-n^{-q}$ 的概率对 $q \geq 1$ 成立，我们就称该界以高概率（***w.h.p.***）成立，其中 n 是输入规模。

平均连接、单连接、质心连接和中位数连接。已有多篇论文研究并行 HAC，例如采用单连接度量的并行 HAC [22], [23], [26]–[28]，以及采用其他度量的并行 HAC [22]。Song 等人 [13] 表明，DBHT 与 PMFG 结合的表现优于完全连接和平均连接 HAC。我们也在第七节表明，我们的方法优于完全连接和平均连接 HAC。Musmeci 等人 [6] 表明，在股票数据集上，DBHT 与 PMFG 结合比单连接、平均连接、完全连接和 k-中位数方法产生的簇质量更高。

其他层次聚类方法也已有相关研究，例如划分式层次聚类算法、结合凝聚式方法与划分式方法的算法 [29], [30]，以及基于密度的方法 [23], [31]。与基于密度的方法 [23], [31],相比，我们的算法有两项优势：首先，许多基于密度的方法最适用于低维数据，而我们的算法不受此限制。此外，这些聚类方法需要一些先验信息来设置参数，以获得高质量的簇。相比之下，DBHT 不需要任何额外参数。Dash 等人 [29]提出了一种并行层次聚类算法，该算法将数据划分为重叠单元，并对这些单元进行聚类。然而，其并行版本仅针对低维数据集进行了描述。Cohen-Addad 等人 [32]设计了一种用于层次化 k-中值问题的并行算法，但该算法仅适用于欧几里得度量。此外，还有一些针对特定类型数据设计的算法，例如针对分类数据 [33] 和基因组数据 [30]的算法。虽然大多数并行算法都是为共享内存设计的，但也有一些算法是为分布式内存设计的 [34]。

PMFG 是加权极大平面图（WMPG）问题的一种近似解，该问题是 NP困难 [12]。此外，也有其他针对 WMPG 问题设计的近似解法。Massara 等人 [5] 提出了 TMFG，该方法基于局部拓扑移动。Kataki 等人 [35] 设计了一种启发式算法，该算法基于向连通生成子图问题和对偶图的转换。Osman 等人 [36] 使用贪心随机自适应搜索过程来求解 WMPG 问题。针对 WMPG 问题还提出了其他启发式算法 [37]–[39]。

除了基于拓扑约束过滤图之外，还有其他用于聚类的图过滤方法。例如，Ma 等人 [40] 使用低通滤波器提取适用于子空间聚类的数据表示，而 Tremblay 等人 [41] 使用图滤波来加速谱聚类。另一种用于聚类的图构建算法是 b-匹配 [42]–[44]。b-匹配算法会过滤图边，使每个节点的入度和出度都不超过 b，并且可以在多项式时间内求解。

IV. 并行三角最大过滤图

本节介绍我们用于并行化 TMFG 构建的算法，其伪代码如算法 1 所示。我们首先进行高层次概述。我们的并行算法会同时向图中添加多个顶点。添加

算法 1： 并行三角最大过滤图

输入： $n \times n$ 相似度矩阵 S，前缀大小 PREFIX ≥ 1

1 // 在 {v1} 中，其行总和最高的四个顶点行总和
$\mathcal{C} = \{v_1, v_2, v_3, v_4\}$
2 $\mathcal{E} = \{(v_1, v_2)(v_1, v_3),(v_1, v_4),(v_2, v_3), (v_2, v_4), (v_3, v_4)\}$
3 // 初始图中的四个三角面图
$\mathcal{F} = \{\{v_1, v_2, v_3\}, \{v_1, v_2, v_4\}, \{v_1, v_3, v_4\}, \{v_2, v_3, v_4\}\}$
4 $V = \{v_5, \ldots, v_n\}$
5 // 每个三角形的最佳顶点 V
增益 $= [\text{argmax}_{u \in V}(\sum_{v \in t} S[v, u]),$ 针对 $t \in \mathcal{F}]$
6 初始化气泡树 T，使用 $\mathcal{C}$
7 外部面 $= \{v_1, v_2, v_3\}$
8 **当** $|V| > 0$ **时**
9 L = 使用并行排序，选择 GAINS 中增益最大的前缀顶点-面配对
10 对于在 L中与多个面配对的顶点，仅保留增益最大的配对，使用并行过滤和并行排序。
11 $V = V \setminus \{\text{顶点} \in L\}$
12 **并行循环** $(v, t = \{v_x, v_y, v_z\}) \in L$ **执行**
13 $\mathcal{E} = \mathcal{E} \cup \{(v, v_x), (v, v_y), (v, v_z)\}$
14 $\mathcal{F} = \mathcal{F} \cup \{\{v, v_x, v_y\}, \{v, v_y, v_z\}, \{v, v_x, v_z\}\} \setminus t$
15 **并行循环** $t \in \{t' :$ 增益$[t'] = v\} \cup \{\{v, v_x, v_y\}, \{v, v_y, v_z\}, \{v, v_x, v_z\}\}$ **执行**
16 增益$[t]$ = argmax $_{u \in V}(\sum_{v \in t} S[v, u])$
17 更新气泡树(v, t, T)
18 **返回** $\mathcal{E}$

多个顶点可能会导致结果与顺序 TMFG 算法不同，因为顺序算法每次插入后都会更新图，使后续顶点有更多面可供选择，而我们的并行算法只有在一轮中的所有顶点都添加完毕后才会更新图。因此，并行添加的顶点越多，结果就越偏离顺序算法。这就在图的质量与可用并行性之间形成了权衡。因此，我们只考虑添加一部分前缀顶点，这些顶点能够带来最高的总边权重增益。前缀大小可以根据应用和可用并行性进行调整。

本节接下来的内容将介绍并行 TMFG 算法（算法 1）的详细信息。输入为一个表示完全无向图的 $n \times n$对称矩阵 S，以及一个参数 PREFIX。高亮显示的代码行用于构建 DBHT，我们将在第五节中进行说明。在第 1–4 行中，我们首先找出 S各行总和最高的四个顶点 $\{v_1, v_2, v_3, v_4\}$，并将它们之间的所有边添加到结果图的边列表 $\mathcal{E}$中。由这四个顶点构成的四个面将添加到面集合 $\mathcal{F}$中。其余顶点位于集合 V中，稍后将添加到图中。在第 5 行中，对于 $\mathcal{F}$中的每个面，我们找出其最佳顶点（即插入该面后能使增益最大化的顶点）。

在第8至18行中，我们分批将其余顶点插入图中，每批最多包含 PREFIX 个顶点。在第9至11行中，我们获取要插入的顶点批次，以及要插入这些顶点的对应面。我们首先从 GAINS 数组中找出增益最大的 PREFIX 个顶点-面配对，并将这些配对存入 L。这可以通过对 GAINS 数组进行并行排序来实现。在第10行中，我们确保每个给定的顶点只会被添加到一个面，以避免竞态条件。对于给定顶点的冲突，通过仅允许该顶点将自身添加到能带来最高增益的面来解决。我们可以使用并行排序，根据哪个面（在所有面中）能带来最高增益对顶点进行分组，并让每个面选出（在所有选择该面的顶点中）能带来最高增益的顶点。

算法 2：更新气泡树

```
1 函数 更新气泡树(v, t = {v_x, v_y, v_z}, T):
2     b* = 新气泡 {v, v_x, v_y, v_z}
3     b = 所在的气泡 t
4     如果 t ≠ 外部表面 则
5         b.父节点 = b*
6         将 b 添加到 b*.CHILDREN
7         外部面 = {v, v_x, v_y}
8     else
9         b*.PARENT = b
10        将 b* 添加到 b.CHILDREN
```

随着图中面数量的增加，可用并行性也会增加。如果 PREFIX $= 1$，则可将第9至10行简化为一次并行最大值计算。在第11行中，我们使用并行筛选从 V 中移除 L 中的顶点。

在第 12–16 行中，我们遍历 L 中的顶点-面配对（v, t），将 v 添加到 t，并更新 $\mathcal{E}$ 和 $\mathcal{F}$。为更新 $\mathcal{F}$，我们添加新生成的三个面，并移除 t。我们还会通过在 V 中为这三个新面以及先前将 v 作为最佳顶点的面计算新的最佳顶点，来更新增益。与原始算法 [5]，遍历所有面以找出先前将 v 作为最佳顶点的面不同，我们为每个顶点维护一个数组，用于存储这类面，从而提高此步骤的效率。

如果我们在算法 1 中选择前缀 $= 1$，则会得到与串行三角最大过滤图算法相同的结果，但第 9 行的并行最大值调用以及第 15–16 行仍能提供一些并行性。这种情况下得到的并行算法与 Massara 等人 [5] 所描述的相同（但他们并未实现）。不过，我们将在第七节中表明，使用前缀 $= 1$ 的并行性有限，因为每轮只能插入一个顶点。

V. TMFG 的并行 DBHT

我们介绍用于构建 DBHT 的并行算法，该算法利用了 TMFG 气泡树的特殊结构。Song 等人 [13]提出的原始 DBHT 构造以一个平面图以及每条边上的两种权重——相似度度量和不相似度度量——为输入。相似度度量来自相似度矩阵 S，而不相似度度量则需要额外提供。构造过程包括以下步骤：从平面图构建无向气泡树 [14]；为气泡树边指定方向；将图顶点分配给气泡；以及使用完全连接聚类生成层次结构。下文将介绍我们如何加速并并行化每个步骤。

A. TMFG 的气泡树

如第二节所述，气泡树 [14] 是一种树，其中节点对应平面子图，而节点之间的边对应原图中将两个相应平面子图分隔开的三角形。具体而言，气泡 是一个极大平面图，其中的三角形均为非分离 [14]。原始 DBHT 构造效率低下，因为它需要先找出图中的所有三角形，然后逐一检验移除每个三角形是否会使图断开。

关键的观察是，可以在 TMFG 构建过程中更高效地构造气泡树，因为在 TMFG 算法中加入的每个顶点都恰好对应一个气泡节点和气泡树中的一条边。在 TMFG 中，每次插入一个新顶点，都会生成一个 4-团；它是一个气泡，因为 4-团的所有面都是非分离三角形。这还会生成一个分离三角形，对应气泡树中的一条新边。这个分离三角形由与其相邻的两个气泡节点所对应的 4-团共享。例如，考虑图 2(a) 中不含顶点 5 的图。如果现在将顶点 5 插入三角形 $t_4 = \{1, 2, 3\}$，我们就创建了一个新的气泡 $b_4 = \{1, 2, 3, 5\}$，而 t_4 现在成为一个分离三角形，由 $b_2 = \{0, 1, 2, 3\}$ 和 b_4共享。我们将气泡树构造整合到并行 TMFG 算法中，从而大幅减少工作量。

我们的 TMFG 无向气泡树具有这样一个不变量：每个气泡节点都有一个父节点，且最多有三个子节点（因为每个 4-团都有一个外面和三个内面）；根节点除外，它没有父节点。此外，一条边的所有后代节点都位于与该边对应的分离三角形的内部一侧。这个不变量对我们加速方向计算至关重要，我们将在后文介绍。

现在我们描述气泡树构造的细节，这些内容在算法 1 中以高亮行标出。第6行，我们用一个气泡节点初始化气泡树，该节点包含我们构造 TMFG 时所用的 4-团。第7行，我们将外面初始化为 $\{v_1, v_2, v_3\}$。在 $\mathcal{C}$ 的四个面中，可以任意选择一个作为外面，因为气泡树的拓扑结构不受此选择影响 [14]。第17行，对于插入到面 t 中的每个顶点 v，我们都会向气泡树添加一个新节点和一条新边 T。这可以并行完成，因为每个顶点 v 都只会插入一个唯一的面 t，因此不会发生冲突。

算法 2 展示了构造气泡树的子程序。第2行，我们创建一个新的气泡节点 b^*。第3行，我们在 T 中找到包含 t 的气泡 b。这个气泡是在创建 t 时创建的（在此前某次调用 UpdateBubbleTree 时）。面 $\{v, v_x, v_y\}$，$\{v, v_y, v_z\}$ 和 $\{v, v_x, v_z\}$ 都位于气泡 b^* 中。如果 t 是当前外面，这意味着我们正将 v 插入外面，而 b 是 T 当前的根节点。因此，在第4–7行，我们令 b 的父节点为 b^*，并将 b 添加到 b^* 的子节点中。这样便保持了上述不变量，因为插入 v 之前，气泡中的顶点位于外面的内部；经过此过程后，这些气泡会成为与外面对应的边 t 的后代节点。随后，将 OUTERFACE 更新为 b^* 对应的 4-团中的一个面 $\{v, v_x, v_y\}$。如果 t 不是当前外面，则无需更改外面，因为该顶点被插入到 b 的某个内面中。在这种情况下，在第8–10行，我们令 b^* 的父节点为 b，并将 b^* 添加到 b 的子节点中。这样便保持了上述不变量，因为 v 必定位于 t 及其祖先节点的内部，而包含 v 的气泡 b^* 是与 t 对应的边的后代节点。

示例 1. 现在，我们通过在图2的示例 TMFG 上运行算法，来演示气泡树构造

。我们将先介绍如何插入单个顶点，然后介绍如何并行插入多个顶点。假设我们从 4-团 $\mathcal{C} = \{0,1,2,4\}$ 开始。我们有四个面$\{\{0,1,2\}, \{0,1,4\}, \{0,2,4\}, \{1,2,4\}\}$，其中 $t_1 = \{0,1,2\}$ 是外面。我们用节点$b_1 = \{0,1,2,4\}$初始化气泡树 T，该节点对应于 $\mathcal{C}$。我们依次将顶点 3、5 和 6 插入面 $\{0,1,2\}$、$\{1,2,3\}$ 和 $\{0,1,3\}$。首先将 3 插入 $t = t_1 = \{0,1,2\}$。对于这次插入，第2行的新气泡 b^* 是 $b_2 = \{0,1,2,3\}$，而第3行的 b 是 b_1。由于 t 是当前的外面，我们令 b_2 成为 b_1的父节点。新的外面是 $t_2 = \{0,1,3\}$。现在，假设我们分别将 5 和 6 并行插入 $\{1,2,3\}$ 和 $\{0,1,3\}$。这与插入 3 类似，我们并行添加 $b_3 = \{0,1,3,6\}$ 作为 b_2的父节点，并添加 $b_4 = \{1,2,3,5\}$ 作为b_2的子节点。

B. 气泡树边定向

现在，我们介绍如何在通过算法1和算法2获得无向气泡树后，对气泡树边进行定向，以及如何显著提高这一步的效率，使其优于原始的顺序 DBHT 算法。气泡树的每条边都对应一个分离三角形。定向方式是计算 TMFG 中连接该三角形与其内部的边权重之和，并将其与连接该三角形与外部的边权重之和进行比较来决定 [13]。图 2(c) 展示了边定向的示例。

我们将指向内部的边权重之和记为 INVAL，将指向外部的边权重之和记为 OUTVAL。在原始算法中，对于每个分离三角形t，都要在 $G\backslash t$ 上运行广度优先搜索（BFS），以找出其外部和内部，然后计算边权重之和。由于平面最大过滤图和 TMFG 分别有 $\Theta(n)$ 条边 [13]，且每次广度优先搜索需要 $\Theta(n)$ 的工作量，因此这需要 $\Theta(n^2)$ 的工作量。

在我们的气泡树中，分离三角形的内部包含与该分离三角形对应的边的所有后代节点中的顶点，而外部则包含其余所有顶点。利用这一性质，我们可以从气泡树的根节点开始，通过一种新颖的递归算法，以 $\Theta(n)$ 工作量计算所有边的方向。在每个气泡节点 b 处，我们计算从该节点指向其父节点的边的方向，并递归地对气泡的子节点调用该过程。与原始算法的二次复杂度工作量相比，这带来了显著改进。

具体而言，我们通过递归调用计算方向函数来计算所有边的方向，该函数见算法 3；初次调用时以气泡树的根节点作为参数。在每个气泡节点 b 处，我们计算从该节点指向其父节点的边的方向。如果 b 是根节点，那么它没有指向父节点的边，因此除了初始化对子节点的计算外，无需进行其他计算（第 20–22 行）。否则，我们计算气泡节点到其父节点的边所对应的分离三角形的 INVAL 和 OUTVAL。如果 INVAL > OUTVAL，那么边的方向是从 b 的父节点指向 b，反之亦然（第 14–17 行）。在第 3–4 行

，我们得到顶点 $\{v_x, v_y, v_z\}$在

算法 3： 计算方向

```
1  函数 计算方向(气泡树节点 b):
2      如果 b 有父节点 则
3          {v_x, v_y, v_z} = b 与其父节点共有的顶点
4          v = b \ {v_x, v_y, v_z}
5          r = {}
6          r[v_x] = w(v_x, v) r[v_y] = w(v_y, v) r[v_z] = w(v_z, v)
7          并行循环 b* ∈ b的子节点 执行
8              r* = {v*_x : val*_x  v*_y : val*_y, v*_z : val*_z} =
                 计算方向(b*)
9              如果 v*_x ∈ r 则 WRITEADD(r[v*_x], val*_x)
10             if v*_y ∈ r 则 WRITEADD(r[v*_y], val*_y)
11             如果 v*_z ∈ r 则 WRITEADD(r[v*_z], val*_z)
12         INVAL = r[v_x] + r[v_y] + r[v_z]
13         OUTVAL = deg(v_x) + deg(v_y) + deg(v_z) − INVAL −
             2(w(v_x, v_y) + w(v_x, v_z) + w(v_y, v_z))
14         如果 无效输 > 无效 则
15             定向边：从 b的父节点指向 b
16         else
17             定向边：从 b指向 b的父节点
18         返回 r
19     否则 /* b 是根节点 */
20         对 b* ∈ b的子节点 并行循环
21             计算方向(b*)
22         返回 {}
```

算法 4： 用于 TMFG 的并行 DBHT

```
   输入： n × n 差异度矩阵 D、加权无向平面图 G
          由 TMFG 生成，气泡树 T
1  计算 T 的根节点的方向
2  v.g = (−∞, −∞) v.q = (−∞, −∞) ∀ 顶点 v ∈ G
3  BB = 包含 T 的节点的集合
4  CVGBB = {b ∈ BB: b.出度=0}
5  对 b ∈ BB 执行并行循环
6      广度优先搜索(T, b)
   // 全源最短路径
7  A = allPairsShortestPaths(G, D)
   // 将顶点分配到汇聚气泡
8  并行循环 b ∈ CVGBB 执行
9      并行循环 v ∈ b 执行
10         χ = computeChi(v, b)
11         WRITEMAX(v.g, (χ,b))
12 V_b^0 = {v ∈ G : v.g = b} ∀b
13 v.g = (∞, ∞)，如果 v.g 是 (−∞, −∞) ∀v
14 并行循环 b ∈ CVGBB 执行
15 并行循环 v ⇌ b 和 v.g == (∞, ∞) 执行
16     L̄ = 计算平均最短路径(v, V_b^0)
17     WRITEMIN(v.g, (L̄, b))
   // 将顶点分配到气泡
18 并行循环 b ∈ BB 执行
19     χ_total = 0
20     对于 v ∈ b 执行
21         χ_v = 计算卡方(v, b); χ_total += χ_v
22     对于 v ∈ b 执行
23         WRITEMAX(v.q, (χ_v / χ_total, b))
   // 使用完全连接构建层次结构
24 {Z_1, ..., Z_n} /* 初始化树状图节点 */
25 并行循环b^c ∈ CVGBB 执行
26     BB_{b^c} = {v.q : v ∈ b^c}
27     并行循环 b ∈ BB_{b^c} 执行
28         令 Z_{(b^c,b)} = completeLinkage({Z_v : v.q = b ∧ v.g = b^c})
29 并行循环b^c ∈ CVGBB 执行
30     令 Z_{b^c} =完全链接({Z_{(b^{c'}, *)} : b^{c'} = b^c})
31 Z = 完全链接({Z_{b^c} : b^c ∈ CVGBB})
32 树状图 = 计算高度(Z)
33 返回树状图
```

由从 b 到其父节点的边表示的分离三角形，并令 v 为气泡中的剩余顶点。在第5–6行，我们用三角形各顶点到 v的边权重，初始化从三角形各顶点到内部的边权重之和，因为 v 位于内部。然后，在第7–11行，我们以递归且并行的方式，计算从 b的子节点的顶点到子节点内部的边权重之和。请注意，这种计算属于嵌套并行，也就是说，会递归地创建并行任务。由于子节点的内部也是b的内部，我们可以将从其子节点获得的 b各顶点的权重相加，以计算 b的 INVAL。在第13行，我们通过从三角形各顶点的加权度中减去 INVAL 和三角形的边

权重，计算 OUTVAL。在第18行，我们将各顶点处的权重返回给 b的父节点。

现在，我们描述算法 3 第13行公式的推导。令 $D = deg(v_x) + deg(v_y) + deg(v_z)$ 表示 v_k 到内部的总加权度，令 O_k 表示 v_k 到外部的总加权度。我们可以将 D 重写如下：

$$\begin{aligned} D &= I_x + O_x + w(v_x, v_y) + w(v_x, v_z) \\ &\quad + I_y + O_y + w(v_y, v_x) + w(v_y, v_z) \\ &\quad + I_z + O_z + w(v_z, v_x) + w(v_z, v_y) \\ &= (I_x + I_y + I_z) + (O_x + O_y + O_z) \\ &\quad + 2(w(v_x, v_y) + w(v_x, v_z) + w(v_y, v_z)) \\ &= \text{INVAL} + \text{OUTVAL} + 2(w(v_x, v_y) + w(v_x, v_z) + w(v_y, v_z)) \end{aligned}$$

重新排列可得到用于计算 OUTVAL 的以下公式：

$$\text{OUTVAL} = D - \text{INVAL} - 2(w(v_x, v_y) + w(v_x, v_z) + w(v_y, v_z))$$

示例2。我们继续介绍算法 3 的示例。给定图2(b)中的有根气泡树，我们从根节点 b_3 开始计算。由于 b_3 没有父节点，我们递归向下处理其子节点 b_2。对于 $b = b_2$，与其父节点共享的顶点是 $\{v_x, v_y, v_z\} = t_2 = \{0, 1, 3\}$，剩余顶点是 $v = 2$。然后，我们在第6行初始化 $r[0] = w(0, 2)$、$r[1] = w(1, 2)$ 和 $r[3] = w(3, 2)$。接下来，我们递归向下处理 b_2 的子节点 b_1 和 b_4。对于 b_1，所得 r^* 数组应包含 $r^*[0] = w(0, 4)$、$r^*[1] = w(1, 4)$ 和 $r^*[2] = w(2, 4)$，这是在第8行递归得到的，因为与其父节点共享的顶点是 $t_1 = \{0, 1, 2\}$，剩余顶点是 4。在第9–11行，由于 $v_x^* = 0 \in r$ 和 $v_y^* = 1 \in r$，我们将 $r[0]$ 增加 $w(0, 4)$，并将 $r[1]$ 增加 $w(1, 4)$。由于 $v_z^* = 2 \notin r$，我们不对其进行处理。现在，$r[0] = w(0,4) + w(0, 2)$，$r[1] = w(1, 4) + w(1, 2)$。同样，对于 b_4，我们将 $r[1]$ 增加 $w(1, 5)$，并将 $r[3]$ 增加 $w(3, 5)$。现在，$r[0] = w(0, 4) + w(0, 2)$，$r[1] = w(1, 4) + w(1,2) + w(1, 5)$，以及 $r[3] = w(3, 2) + w(3, 5)$。当我们在第12行处理 b_2 时，$r[0]$、$r[1]$ 和 $r[3]$ 的和为 INVAL，因为它包含从 t_2 的三个顶点到其内部的边权重。OUTVAL 的计算方式是将从 t_2 出发的所有边的权重相加，然后减去 INVAL 以及 t_2 的边权重（每个方向各计算一次）。在我们的示例中，我们发现 INVAL $>$ OUTVAL，因此该边的方向为从 b_3 指向 b_2（图2(c)）。通过比较 INVAL 和 OUTVAL，可以类似地确定其他边的方向。

C. 顶点分配

在本小节中，我们首先介绍 DBHT 算法的分配规则，然后介绍用于计算分配的并行算法。汇聚气泡 是只有入边、没有出边的气泡。直观地说，汇聚气泡是“沿着最强连接的有向路径的终点” [13]，因此被视为局部簇的中心。第一层聚类会将每个顶点分配给一个唯一的汇聚气泡。如果一个顶点至少属于一个汇聚气泡，那么它会被分配给连接度最强的汇聚气泡 $\chi(v, b) = \frac{\sum_{u \in b} w(u, v)}{3(|b|-2)}$，其中 ~~$3(|b|-2)$~~ 是该气泡中的边数 b。对于 TMFG，所有气泡都有 6 条边，因此可以将其简化为 $\chi(v, b) = \sum_{u \in b} w(u, v)$。对于不属于任何汇聚气泡的顶点，则将其分配给平均最短路径距离最小的汇聚气泡：

$$\bar{L}(v, b) = mean\{l_D(u, v) \mid u \in V_b^0 \wedge v \rightharpoonup b\}.$$

$v \rightharpoonup b$ 表示 v 位于有向气泡树中能够到达 b 的某个气泡中，而 $l_D(u, v)$是使用不相似度度量D 作为边权重时，TMFG 中 u 到 v 的最短路径距离。对于所有气泡 b，我们令 V_b^0为已通过计算 χ分配给 b的汇聚气泡中的顶点集合。在上述过程中分配给同一汇聚气泡 b 的顶点构成一个组。

在完成顶点的初始划分后，我们通过进行第二层聚类来考察这些组各自的内部结构。这一次，我们将每个顶点分配给唯一的气泡，但该气泡不一定是汇聚气泡。顶点 v 被分配给使另一种连接度分数达到最大值的气泡 b

$$\chi'(v, b) = \frac{\sum_{u \in b} w(u, v)}{\sum_{u', v' \in b} w(u', v')}.$$

我们并行 DBHT 算法的伪代码见算法 4。在第 1–23 行中，我们展示如何计算这两种分配。我们首先计算顶点分配计算所需的辅助数据。第 1 行中，我们使用算法 3 计算气泡树边的方向。第 2 行中，我们将每个顶点的字段 g 和 q 初始化为$(-\infty, -\infty)$，以便对它们执行 WRITEMAX 操作。g 是组分配，q 是气泡分配。第 3 行中，我们初始化包含气泡节点的集合 BB。第 4 行中，我们根据树节点的出度，通过并行过滤获取所有汇聚气泡 CVGBB。第 5–6 行中，我们针对每个气泡并行地在有向气泡树中运行广度优先搜索，并记录气泡中的顶点能够到达哪些汇聚气泡。这有助于我们稍后进行 $v \rightharpoonup b$ 计算。第 7 行中，我们从每个气泡节点并行运行迪杰斯特拉最短路径算法，以计算 TMFG 中的全源最短路径。

现在我们已经获得所有必要的辅助数据，接下来开始计算分配结果。在第8–17行，我们计算顶点到汇聚气泡（即组）的顶点分配。首先，我们通过并行计算所有连接度分数 χ），并使用并发的 WRITEMAX 操作写入组分配（第8–11行），来计算至少属于一个汇聚气泡的顶点的组分配。在第12行，我们为所有汇聚气泡 b 计算 V_b^0：先使用并行整数排序，按组分配对顶点进行排序，然后使用并行筛选，获取排序后的顶点数组中各组的起始和结束索引。在第13行，我们为所有组 v 尚未分配的顶点 g 设置 $v.g = (\infty, \infty)$，以便执行 WRITEMIN 操作。然后，我们并行遍历所有汇聚气泡 b 和所有顶点

v，使得 $v \rightharpoonup b$ 且 v 尚未分配给任何汇聚气泡。在循环中，我们计算平均平均最短路径距离 L，并使用 WRITEMIN 将最小 L 的汇聚气泡写入每个顶点的分配结果。接下来，在第18–23行，我们计算气泡分配。这与第8–11行类似，但我们使用变量χ_{total}记录气泡中的边权重总和，并在寻找连接度最大的气泡时，使用 χ_{total} 对连接度重新加权。[2]

示例 3. 我们继续看图 2(c) 中的示例。这里我们只有一个汇聚气泡 b_2，因为它是唯一一个没有出边的气泡。因此，在第 8–17 行，所有顶点都被分配到这个汇聚气泡。接下来，我们看看第 18–23 行会发生什么。顶点 4、5 和 6 只属于一个气泡，因此被分配到它们所在的气泡。顶点 0、1、2 和 3 属于多个气泡，因此我们计算 χ' 分数，并将每个顶点分配到 χ' 最大的气泡。例如，对于顶点 3，我们将计算 $\chi'(3, b_2)$、$\chi'(3, b_3)$ 和 $\chi'(3, b_4)$。在这个示例中，我们发现 $\chi'(3, b_3)$ 最大，因此将顶点 3 分配到 b_3。

D. 完全链接

现在我们描述如何从顶点分配中获得树状图。概括而言，我们在三个层次上构建完全连接层次结构：气泡内、气泡间和组间 [13]。对于完全连接算法，两个顶点 u 和 v 之间的距离是它们的最短路径距离 $l_D(u, v)$，两个顶点集合V_1 和 V_2 之间的距离则为 $d(V_1, V_2) = \max\{l_D(u, v) \mid u \in V_1, v \in V_2\}$。我们使用 Yu 等人提出的并行完全连接算法 [22]。

构建树状图的步骤见算法 4 的第 24–33 行。在第24行，我们初始化 n个树状图节点，TMFG 中的每个顶点对应一个。我们将子组 定义为组中属于同一气泡的顶点集合。在第25–28行，对于每个子组，我们对该子组内各顶点对应的树状图节点运行完全连接算法。结果是每个子组各自得到一棵树状图。我们使用子组，是为了避免将同一组中但属于不同气泡的顶点一起处理，这与顺序算法 [13]一致。接着，在第30行，我们对每个组中的树状图节点运行完全连接算法，得到每个组各自的树状图。最后，在第31行，我们对各组的树状图节点运行完全连接算法，以获得最终的树状图。

树状图高度。 在传统的完全连接聚类中，树状图高度是两个合并簇之间的距离。树状图的高度要求，靠近树状图根节点（较晚合并的簇）的节点，其高度至少要与远离根节点的节点高度相同。不过，由于我们将三个层级的树状图串联起来，最短路径距离可能不满足高度要求。例如，气泡内两个顶点之间的最大最短路径距离，可能大于两个气泡之间的距离。因此，我们必须重新指定树状图节点的高度。我们在实现中采用与 Aste [45],相同的高度指定方法，以确保所有恰好包含一个组的节点高度相同。

对于在第31行合并的组间树状图节点，其高度为其后代节点中汇聚气泡的数量。这可以通过从根节点 $\mathcal{Z}$开始自顶向下遍历来计算。属于同一汇聚气泡 b^c 的气泡内和气泡间树状图节点，其高度从$[\frac{1}{n_b-1}, \frac{1}{n_b-2}, \ldots, \frac{1}{2}, 1]$中选取，其中 n_b是分配给汇聚气泡 b^c的顶点数量。每个气泡的树状图节点在高度列表中占据一个连续区段。为了获得每个$\mathcal{Z}_{(b^c)}$后代节点的树状图高度，我们对这些树状图节点进行排序，使在第28行合并的节点排在第30行合并的节点之前。在第30行合并的节点按合并时的距离排序。在第28行合并的节点则先按气泡分配排序，再按合并时的距离排序。现在，同一气泡和同一汇聚气泡中的所有树状图节点都连续排列。然后，我们按排序后的顺序为树状图节点分配高度$[\frac{1}{n_b-1}, \frac{1}{n_b-2}, \ldots, \frac{1}{2}, 1]$。这种排序确保树状图中的组间层次结构位于气泡间层次结构之下。

示例 4。 在图 2(c) 中，由于 $\{2, 4\}$、$\{0, 3, 6\}$ 和 $\{1, 5\}$ 都被分配到同一个气泡中，我们首先得到三个树状图 $\mathcal{Z}_{(b_2,b_1)}$、$\mathcal{Z}_{(b_2,b_3)}$ 和 $\mathcal{Z}_{(b_2,b_4)}$。然后，对这三个树状图根节点运行完全连接，得到一个单一的树状图 $\mathcal{Z}_{b_2}$。在本例中，我们只有一个汇聚气泡，因此处理完成。如果存在多个汇聚气泡，我们的算法将在组树状图根节点上运行完全连接；这些根节点的高度都为 1，从而生成最终的树状图。所得树状图如图 2(d) 所示。

VI. 分析

本节使用第 II 节中定义的工作-跨度模型，分析我们的并行算法的复杂度。

我们用 ρ 表示并行 TMFG 算法所需的轮数，即在算法 1 的第8行迭代 while 循环的次数。我们用 π 表示 PREFIX 的大小。

并行 TMFG。 首先，我们分析第四节中给出的并行 TMFG 算法的复杂度。计算初始四个顶点的第一步需要 $O(n^2)$ 工作量，跨度为 $O(1)$。初始化 GAINS 数组需要 $O(n)$ 工作量，跨度为$O(1)$，因为初始时 V中有 $n-4$ 个节点，且 $\mathcal{F}$ 中有四个面。

现在我们分析 while 循环。第 9 行的并行排序需要 $O(n\log\ n)$ 工作量和 $O(\log n)$ 跨度。第 10–11 行的并行过滤和排序需要 $O(\pi\ \log\ \pi)$ 时间和 $O(\log \pi)$ 跨度。在 for 循环中，L 中的每个顶点在第 13–14 行贡献 $O(1)$ 工作量和跨度。在第 15–16 行，我们维护一个

[2]在他们的论文中， [13], Song 等人在第2行至第11行将分配给某个汇聚气泡的顶点保留在该气泡中，不会将其重新分配给其他气泡。然而，他们的实现确实会重新分配，我们选择与他们的实现保持一致。

每个面对应的顶点排序列表。每次创建新面时，我们需要花费 $O(n\log n)$ 工作量，根据顶点相对于该面的增益对所有顶点排序，并为每个面维护一个已排序顶点链表。每个顶点都存储指向其在这些链表中所在位置的指针。当我们将一个顶点插入某个面时，会将该顶点从所有其他面的链表中移除。由于共有 $O(n)$ 个面，排序的总工作量为 $O(n^2 \log n)$，从链表中移除顶点的总工作量为 $O(n^2)$，因为每个顶点在每个面对应的链表中恰好移除一次。该算法的跨度为 $O(\rho\log n)$，因为每轮的跨度为 $O(\log n)$。

气泡树构造。 算法 1 中高亮的代码行（调用算法 2）构造了气泡树。构建气泡树需要 $O(n)$ 工作量，因为我们插入 n 个顶点，且每次插入需要 $O(1)$ 工作量。由于每轮都以 $O(1)$ 跨度更新气泡树，总跨度为 $O(\rho)$。

边定向。 算法 3 计算气泡树边的方向。计算气泡树的边方向需要 $\Theta(n)$ 工作量，因为每个气泡需要 $O(1)$ 工作量（和跨度），而气泡总数为 $n-3$。计算边方向的跨度在渐近意义上与气泡树的高度相同。这是因为在算法 1 的每一轮中，我们只添加与现有气泡节点相邻的树边，因此高度每轮最多增加 2，该步骤的跨度因而为 $O(\rho)$。

顶点分配。 现在，我们给出并行 DBHT 算法中顶点分配步骤（算法 4 的第 4–23 行）的复杂度。第 4、8–13 和 17–23 行需要 $O(n)$ 工作量，因为共有 $O(n)$ 个气泡，且每个气泡有四个顶点。第 14–17 行需要 $O(n^2)$ 工作量，因为汇聚气泡至多有 $O(n)$ 个，且在最坏情况下，每个顶点最多可以到达 $O(n)$ 个汇聚气泡。第 8–23 行的跨度为 $O(1)$，但第 12 行除外；该行因整数排序而有 $O(\log n)$ 的跨度。第 5–6 行需要 $O(n^2)$ 工作量，因为我们运行了 $O(n)$ 次广度优先搜索。运行一次广度优先搜索的跨度为 $O(\rho\log n)$，因为气泡树的直径为 $O(\rho)$ [46]。

第 7 行上的全源最短路径 (APSP) 可以通过从所有节点运行单源最短路径 (SSSP) 来计算，也可以通过专门的 APSP 算法来计算。无论是在顺序还是并行设置下，APSP 和 SSSP 问题都已得到充分研究 [47]–[56]。此外，还有专为平面图设计的 APSP 和 SSSP 算法，其复杂度优于一般情形下的对应算法 [57]–[69]。Federickson [59] 给出了一个顺序算法，可在 $O(n^2)$内计算 APSP，但我们尚未发现任何能以更低深度实现相同工作量的并行算法。无向平面图上的 APSP 算法示例包括 Pan 和 Reif 提出的一种算法，其工作量为$O(n^2 \log n)$，跨度为 $O(\log^2 n)$ [70], ; 以及 Henzinger 等人提出的一种算法，其工作量为 $O(n^{7/3}\log\, n\log(nD))$，跨度为 $O(n^{2/3}\log^{7/3} n\log(nD))$，其中 D 是边权重绝对值之和 [57]。设所用 APSP 算法的工作量为 W_{APSP}，跨度为 S_{APSP}。**完全链接**。 最后，我们考虑完全链接这一步。设 ρ_l 为并行算法所用的轮数完全链接算法 [22]。最坏情况下，其跨度为 $O(\rho_l \log n)$，工作量为 $O(\rho_l n^2)$。Yu 等人 [22] 表明，在实际应用中，完全链接算法的运行时间接近 $O(n^2)$。

摘要。 总的来说，TMFG 的工作量为 $O(n^2 \log n)$，跨度为 $O(\rho \log\, n)$；DBHT 的工作量为 $O(W_{APSP} + \rho_l n^2)$，跨度为 $O(S_{APSP} + \rho + \rho_l \log\, n)$。与 Song 等人相比， [13], 我们的算法在所有步骤中的跨度都更短，并且在气泡树构造和方向计算中的工作量更少。Song 等人在气泡树构造和方向计算中的工作量为 $O(n^2)$，而我们的算法仅为 $O(n)$。我们将在第七节展示，我们的所有步骤都更快。他们的 DBHT 代码中最快的部分是全源最短路径计算，其他部分的运行时间则慢了几个数量级；我们显著缩短了 DBHT 代码所有部分的运行时间，现在的瓶颈是全源最短路径计算。

VII. 实验

测试环境。 我们在 Amazon EC2 上的 c5.24xlarge 机器上进行实验，该机器配备 2 个 Intel Xeon Platinum 8275CL（3.00GHz）中央处理器，共有 48 个启用超线程的核心，内存为 192 GB。默认情况下，我们使用所有启用超线程的核心。对于经过测试的 C++ 代码，我们使用 g++ 编译器（版本 7.5）并启用 -O3 标志，同时使用 ParlayLib [71]实现代码中的并行性。

- **PMFG-DBHT** 是现有的 MATLAB 串行 PMFG 和 DBHT 实现，我们从网上获取了该实现 [45]。PMFG-DBHT 的瓶颈在于构建 PMFG。我们尝试用 C++ 实现相同的 PMFG 构建算法，但发现其速度比 MATLAB 实现更慢，因此报告 MATLAB 的运行时间。
- **SEQ-TDBHT** 是 MATLAB 中最先进的串行 TMFG 和 DBHT 实现，我们从网上获取了该实现 [45]。
- **PAR-TDBHT** 是我们用 C++ 实现的并行 TMFG 和 DBHT。我们测试了大小为 1、2、5、10、30、50 和 200 的前缀。
- **COMP** 和 **AVG** 分别是 Yu 等人 [22]实现的并行 C++ 完全连接和平均连接实现。
- **K-MEANS** 是用 C 编写的可扩展 k-means++ [72] 实现 [73]，并使用 OpenMPI 进行了并行化。
- **K-MEANS-S** 是 scikit-learn 中并行 k-means++ 的实现，包含一个预处理步骤，用于计算谱嵌入；该步骤通过近邻图 [74] 构建相似度矩阵。我们选择能获得最佳聚类质量的最近邻数量。如果真实簇数为 c，则将原始数据投影到 c 维空间。我们发现，这是找到优质簇的一种有效启发式方法。我们还尝试了使用 Eigen [75], 的 C++ 实现，但发现 scikit-learn 版本速度更快。

评估。 我们使用调整兰德指数（ARI） [11] 和调整互信息（AMI） [76]分数来评估聚类质量。在我们的实验中，AMI 与 ARI 呈现出相似的趋势，因此我们仅展示 ARI 的图表。

表 II： 实验中使用的 UCR 数据集概要。 n 是对象数量， L 是对象的长度或大小。

ID	Name	n	L	类别数
1	Mallat	2400	1024	8
2	UWaveGestureLibraryAll	4478	945	8
3	NonInvasiveFetalECGThorax2	3765	750	42
4	MixedShapesRegularTrain	2925	1024	5
5	MixedShapesSmallTrain	2525	1024	5
6	ECG5000	5000	140	5
7	NonInvasiveFetalECGThorax1	3765	750	42
8	StarLightCurves	9236	84	2
9	HandOutlines	1370	2709	2
10	UWave 手势库 X	4478	315	8
11	CBF	930	128	3
12	昆虫翅膀拍动声	2200	256	11
13	UWave 手势库 Y	4478	315	8
14	ShapesAll	1200	512	60
15	SonyAIBORobotSurface2	980	65	2
16	FreezerSmallTrain	2878	301	2
17	Crop	19412	46	24
18	电器设备	16160	96	7

设 n_{ij} 为真实聚类 i和算法生成的簇 j中的对象数量， n_{i*} 为 $\sum_j n_{ij}$， n_{*j} n 为 $\sum_i n_{ij}$，而 $\sum_i n_{i*}$为。ARI 的计算方式如下：

$$\frac{\sum_{i,j}\binom{n_{ij}}{2} - [\sum_i \binom{n_{i*}}{2}\sum_j \binom{n_{*j}}{2}]/\binom{n}{2}}{\frac{1}{2}[\sum_i \binom{n_{i*}}{2} + \sum_j \binom{n_{*j}}{2}] - [\sum_i \binom{n_{i*}}{2}\sum_j \binom{n_{*j}}{2}]/\binom{n}{2}}.$$

完美匹配时，ARI 分数为 1；对于随机分配，其期望值为 0。

计算层次化方法的调整兰德指数时，我们会切割树状图，使生成的簇数量与真实聚类的数量相同。对于 k-means 方法，我们将 k 设置为真实聚类的数量。

数据集。我们展示了 UCR 时间序列分类档案中的 18 个数据集的结果 [77], ，其中包括三个最大的作物、电器设备和 StarLightCurves 数据集（我们从这三个数据集中移除了重复点）。这些数据集汇总于表 II 中。由于 UCR 档案是为分类任务而设计的，并非所有数据集都适合聚类。我们选择了两个最大的数据集，以及 16 个 K-MEANS 的 ARI 分数至少为 0。2 的数据集。

我们还使用 Yahoo Finance API 收集了 2013年1月1日至 2019年1月1日期间 1614 只美国股票的每日收盘价（1761 个交易日）。我们使用行业分类基准（ICB）获取真实聚类。

我们使用皮尔逊相关系数 p 作为相似度度量，并使用 $d = \sqrt{2(1-p)}$ 作为不相似度度量 [78]。对于归一化的零均值向量， d 与欧氏距离平方相同。

A. 运行时间

图 3 展示了所有层次聚类算法在各个数据集上的运行时间（上图为串行时间， 下图为并行时间）。

PAR-TDBHT- 1 是使用大小为 1 的前缀的 PAR-TDBHT，而 PAR-TDBHT-10 是使用大小为 10 的前缀的 PAR-TDBHT（我们在大多数实验中选择了这个前缀大小，因为它在速度和聚类质量之间实现了良好的折衷）。由于 PMFG-DBHT 和 SEQ-TDBHT 是顺序算法，因此我们只在上方的图中列出它们。

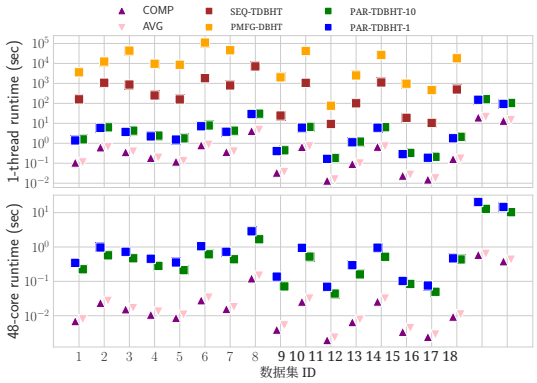


图 3： UCR 数据集上不同方法的运行时间（秒）。上图显示单线程下的运行时间，下图显示使用超线程的 48 个核心上的运行时间。PMFG-DBHT 和 SEQ-TDBHT 在数据集 17 和 18 上超时。PMFG-DBHT 在数据集 8 上也超时。

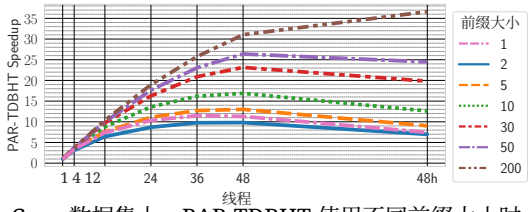


图 4： Crop 数据集上，PAR-TDBHT 使用不同前缀大小时，相对于自身的并行加速比随线程数的变化。”48h”表示使用双路超线程的 48 个核心。由于工作量不足，且超线程带来的开销相对于算法的工作量较大，因此某些数据集的加速比会在启用超线程后下降。

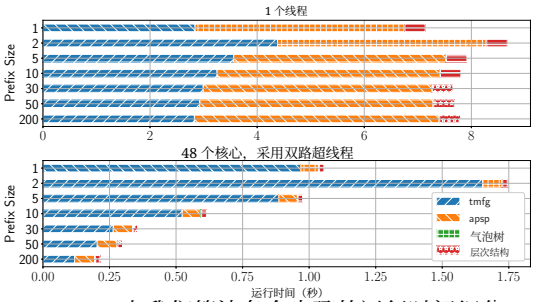


图 5： ECG5000 上我们算法各个步骤的运行时间细分。”tmfg”对应 TMFG 构建（算法 1）；”apsp”对应全点对最短路径计算；”bubble-tree”对应计算气泡树边的方向并将顶点分配到气泡；”hierarchy”对应运行完全连接子程序。

我们可以看到，PMFG-DBHT 和 SEQ-TDBHT 的运行速度都比所有其他方法慢一个数量级以上。单线程下，PMFG-DBHT 比 PAR-TDBHT-1 慢 458–15586 倍，比 PAR-TDBHT-10 慢 414–14254 倍。单线程下，SEQ-TDBHT 比 PAR-TDBHT-1 慢 56–276 倍，比 PAR-TDBHT-10 慢 50–235 倍。在启用超线程的 48 个核心上，SEQ-TDBHT 比 PAR-TDBHT-1 慢 136–2483 倍，比 PAR-TDBHT-10 慢 226–4487 倍。我们将在下文的运行时间分解一节讨论这种加速的原因。PAR-TDBHT-1 和 PAR-TDBHT-10 比 AVG 和 COMP 慢，这在预料之中，因为 DBHT 将完全连接聚类作为一个子程序。不过，我们在第 VII-B 节看到，PAR-TDBHT-1 和 PAR-TDBHT-10 得到的

簇明显优于 AVG 和 COMP，适用于大多数数据集。

不包含 K-MEANS-S 和 K-MEANS，因为它们不会生成树状图，而且要获得不同尺度的簇，就需要多次运行算法。在所有数据集上取平均后，在启用超线程的 48 个核心上，K-MEANS 运行一次比 PAR-TDBHT-1 快 1.82 倍，比 PAR-TDBHT-10 快 1.31 倍；K-MEANS-S 运行一次则比 PAR-TDBHT-1 慢 10.33 倍，比 PAR-TDBHT-10 慢 16.23 倍。K-MEANS-S 使用 12 线程时的运行时间，因为超过 12 个线程后，由于并行开销，速度反而会变慢。虽然我们的算法比 K-MEANS 慢，但 K-MEANS 不是层次化算法，也不具备确定性。

线程数可扩展性。 在图4中，我们展示了算法在最大的 Crop 数据集上随线程数变化的可扩展性。前缀大小为 200 的 PAR-TDBHT 在启用双路超线程的 48 个核心上，实现了 36.6 倍的自相对加速比。总体而言，前缀越大，可扩展性越高，因为在 TMFG 构建过程中可以并行处理更多插入操作。然而，使用大小为 2 的前缀实际上比使用大小为 1 的前缀更慢；后者对应精确 TMFG。原因如下：当前缀大小为 1 时，我们的实现只需通过并行最大值找到要插入的最佳顶点-面片对；但当前缀大小大于 1 时，算法必须先对所有顶点-面配对进行排序，再从排序后的配对中选取一个前缀进行插入。大小为 2 的前缀较小，因此额外的并行性不足以抵消排序开销。在我们的实验中，使用大小为 5 或更大的前缀，其运行时间与使用精确 TMFG 相近或更短。我们最大的自相对加速比出现在 StarLightCurves（数据集 8）上：前缀大小为 200 的 PAR-TDBHT 实现了 41.57 倍的加速比。

数据规模可扩展性。 我们观察到，在我们的数据集上，PAR-TDBHT 的运行时间随数据规模增长，在单线程下近似按 $O(n^{2.22})$ 函数变化，在采用双路超线程的 48 个核心上则近似按 $O(n^{1.79})$ 函数变化 n。并行运行时间的扩展性优于单线程运行时间，因为随着 n增大，我们获得了更高的并行加速比。

运行时间分解。 图5展示了我们的算法在 ECG5000数据集上各个步骤的运行时间分解（不同步骤的说明见图注）。在顺序执行时，大部分运行时间花在“tmfg”和“apsp”步骤上；而在并行执行时，大部分运行时间花在“tmfg”步骤上，尤其是在前缀大小较小时，此时 TMFG 构建的并行性有限。当前缀大小较大时，“tmfg”步骤的运行时间会显著缩短。我们用于 TMFG 的“气泡树”非常快，运行时间短到在图中无法显示。

作为对比，在相同数据集上，SEQ-TDBHT 的 “tmfg” 步骤需要 628 秒，“apsp” 步骤需要 9 秒，“bubble-tree” 步骤需要 69 秒，“hierarchy” 步骤需要 1136 秒。因此，PAR-TDBHT 的所有步骤即使在单线程下也都比 SEQ-TDBHT 快。我们的 “tmfg” 步骤更快，是因为我们使用了经过优化的数据结构来更新增益表，无需遍历所有面。“apsp” 步骤更快，是因为我们使用了不同的图数据结构。SEQ-TDBHT 使用 Boost Graph Library 中的 Johnson 算法，而我们使用了同一库中更快的 Dijkstra 算法。我们也尝试在 SEQ-TDBHT 中使用 Dijkstra 算法，发现 MATLAB 的 Boost Graph Library 中，Dijkstra 算法平均比 Johnson 算法慢 16%（这可能是因为需要多次将 MATLAB 图转换为 C++ 格式）。我们的 “bubble-tree” 步骤更快，是因为我们针对 TMFG 拓扑对其进行了优化。尽管在 SEQ-TDBHT 中，这一步比 “apsp” 慢得多，但在 PAR-TDBHT 中，其耗时变得微不足道。我们的 “hierarchy” 步骤更快，是因为我们使用了 Yu 等人的优化完全链接算法， [22], 而 SEQ-TDBHT 使用的是效率较低的实现。在所有数据集上，我们的 “tmfg” 步骤比 SEQ-TDBHT 快 15.62–9629 倍；在 48 个核心上启用超线程时，其余步骤合计比 SEQ-TDBHT 快 1278–14049 倍。在单线程下，我们的 “tmfg” 步骤比 SEQ-TDBHT 快 12.19–319.56 倍，其余步骤合计比 SEQ-TDBHT 快 59.42–431.17 倍。在 PAR-TDBHT 中，顺序运行时，TMFG 和全源最短路径占据了大部分执行时间，这意味着采用更复杂的全源最短路径实现有望进一步缩短运行时间。然而，在基线实现中，[13], 全源最短路径只占运行时间的一小部分，瓶颈在于其他步骤。这意味着我们的算法已经缓解了原始算法的瓶颈。

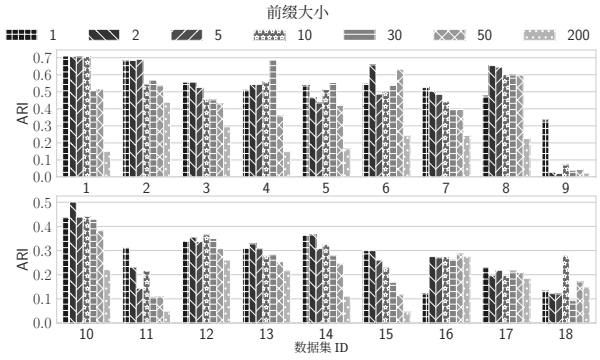


图 6： PAR-TDBHT 的聚类质量（ARI 分数）。不同深浅的颜色表示不同的前缀大小。

B. 聚类质量

前缀大小与聚类质量。 我们的基于前缀的 TMFG 算法能够生成权重非常接近串行三角最大过滤图和 PMFG 算法的过滤图，有时甚至能生成权重更高的过滤图。在我们的实验中，基于前缀的 TMFG 算法生成的图，其边权重总和达到 PMFG 所生成图的边权重总和的 92.1–100.3%。如果只考虑前缀大小不超过 50 的情况，那么边权重总和达到 PMFG 所生成图的边权重总和的 97.1–100.3%。图 7 展示了不同前缀大小对应的边权重总和。

图 6 展示了使用不同前缀大小时的调整兰德指数。我们发现，PAR-TDBHT 使用大小大于 1 的前缀时，得到的调整兰德指数与使用大小为 1 的前缀时相近，有时甚至更高（后者对应于使用精确 TMFG）。

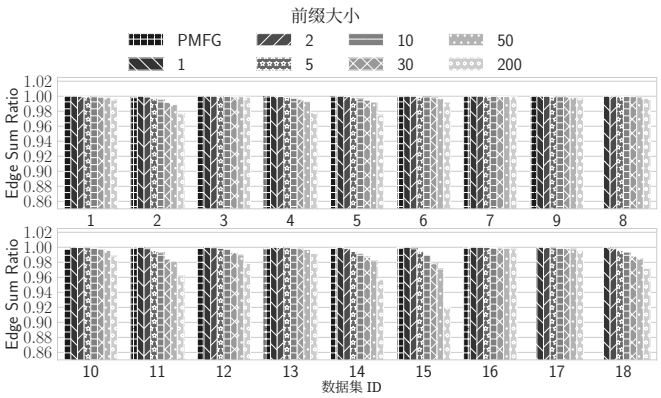


图 7： 边权重总和与 SEQ-TMFG 的边权重总和之比。

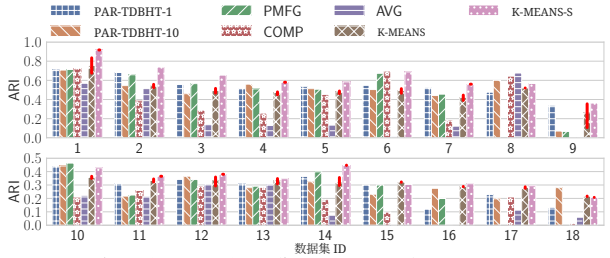


图 8： 不同方法在 UCR 数据集上的聚类质量。由于 COMP 和 AVG 的 ARI 分数接近 0，因此有些柱状条难以观察到。PMFG 在数据集 8、17 和 18 上超时。K-MEANS 的误差线显示了使用不同线程数运行时获得的 ARI 分数范围。

通常，前缀大小越大，调整兰德指数越低；但有时，较大的前缀大小也能提升质量，因为簇对噪声的敏感度可能会降低。我们在附录中给出了这一现象的具体示例。对于较小的数据集（例如数据集 9、11 和 15），调整兰德指数的下降幅度更大。这是因为前缀占过滤图中所有边的比例更高。对于较大的数据集（例如数据集 2、6、8、10、13、17 和 18），调整兰德指数的下降幅度较小。

层次化方法。图 8 展示了所有方法在各个数据集上的聚类质量。我们发现，PAR-TDBHT-1 和 PAR-TDBHT-10 通常比另外两种层次聚类算法 COMP 和 AVG 生成质量更高的簇。在真实聚类数量很少的数据集上，例如数据集 9、15 和 16，COMP 和 AVG 的 ARI 分数较低。这是因为 COMP 和 AVG 对凝聚决策很敏感，而这些决策仅利用局部信息；一旦决策有误，就会导致较低的 ARI 分数。另一方面，DBHT 的拓扑约束（气泡和收敛气泡）利用了全局信息，因此对错误凝聚决策的敏感度较低。当全局信息未被正确捕获时（例如数据集 9），DBHT 也会受到凝聚决策敏感性的影响。

***k*-Means。** PAR-TDBHT-1 和 PAR-TDBHT-10 在所有数据集上生成的簇质量都与 K-MEANS 相近。然而，K-MEANS 不会生成树状图。若要找出不同尺度下的簇，就必须多次运行该算法，这会导致运行时间更长。

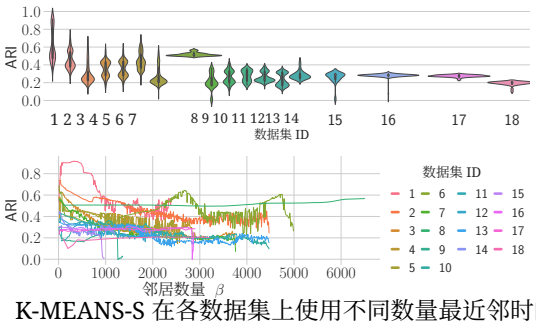


图 9： K-MEANS-S 在各数据集上使用不同数量最近邻时的 ARI。上图展示了不同 β 值对应的 ARI 分数分布。下图展示了 ARI 分数随 β 值的变化，体现出其振荡行为。

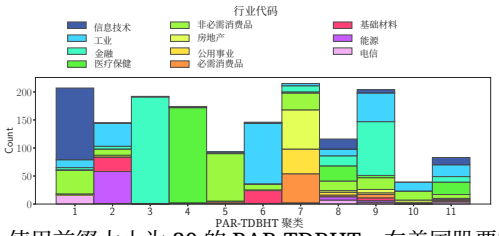
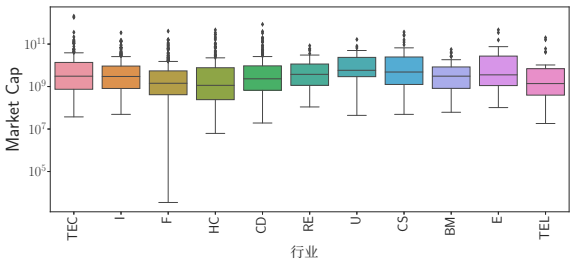


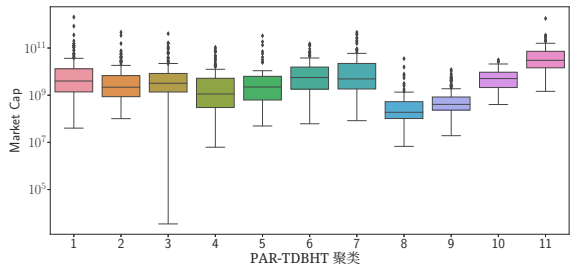
图 10： 使用前缀大小为 30 的 PAR-TDBHT，在美国股票数据集上的聚类结果与真实情况的对比。

使用谱嵌入预处理步骤可提升 K-MEANS 方法，使其在大多数数据集上达到最佳质量。然而，我们在图9中展示了，在许多数据集上，K-MEANS-S 的质量对参数 β（最近邻的数量）的选择高度敏感。在图9的上图中可以看到，对于大多数数据集，使用不同的 β 会产生范围很广的 ARI 分数。在下图中可以看到，对于许多数据集，ARI 会随不同的 β 而振荡，并且最佳的 β 对于每个数据集都大不相同。因此，该参数很难事先选择。我们还在嵌入后的数据集上运行了 PAR-TDBHT，发现其 ARI 分数与未嵌入数据集的相近。对于数据集 8、17 和 18，使用 β 值分别为 6600、3000 和 4000 进行谱嵌入时，我们耗尽了内存。对于其他所有数据集，我们测试了范围为 10 到 n 的 β。

示例：股票聚类。 图10展示了在美国股票数据集上使用前缀大小为30的 PAR-TDBHT 得到的簇。我们使用 Musmeci 等人提出的方法 [6]，通过计算去趋势化每日对数收益率，对每日股票价格进行预处理。随后，我们对归一化去趋势化每日对数收益率进行谱嵌入。最后，我们计算嵌入数据的皮尔逊相关系数，并在相关矩阵上运行 PAR-TDBHT。我们发现，PAR-TDBHT 能够在股票数据中识别出结构和模式。它准确地将“金融”股票、“医疗保健”股票和“非必需消费品”股票聚为一类。此外，还有一个主要由“信息技术”股票组成的簇，以及一个主要由“工业”股票组成的簇。我们还发现，几乎所有“能源”、“公用事业”、“必需消费品”和“房地产”股票都聚在一起。虽然某些簇中的公司并非都来自 ICB 分类中的同一行业，但它们彼此之间存在共性。



(a)



(b)

图 11： 各板块（a）和各PAR-TDBHT 聚类（b）中股票代码的市值箱线图。市值信息取自雅虎财经。板块以其首字母标记。对应的板块名称见表 III。

表 III： 板块名称缩写。

缩写	Name
TEC	信息技术
I	工业
F	金融
HC	医疗保健
CD	非必需消费品
RE	房地产
U	公用事业
CS	必需消费品
BM	基础材料
E	能源
TEL	电信

例如，簇7同时包含非必需消费品股票和必需消费品股票——许多非必需消费品行业的公司属于餐饮业和批发业，而许多必需消费品行业的公司是食品饮料公司，两者有所关联。再比如，在簇1中，许多不属于科技行业的公司，其大部分业务都在互联网上开展。簇1中属于工业行业的大多数公司都位于“电子和电气设备”板块，[3] 该板块与科技相关。

簇8和簇9比其他簇更为混杂，因为这些簇中的公司规模较小，因此其股价可能更为波动。我们可以从图11中看到，虽然各板块之间的市值中位数没有显著差异（图11(a)），但簇8和簇9的市值总体上低于其他簇（图11(b)）。

该聚类的 ARI 分数为 0.36。相比之下，精确 TMFG 聚类的 ARI 分数为 0.28。

[3]在ICB中，分类分为四个层级：行业、超行业、板块和子板块。

在实际应用中，股票聚类可用于投资组合优化和风险对冲 [6], [79]。

时间与质量权衡。 PAR-TDBHT在运行时间和聚类质量之间实现了良好的权衡。它比K-MEANS-S方法更快，并且能够生成质量相近的簇。虽然PAR-TDBHT比AVG和COMP慢，但其聚类质量更加稳定，而且在大多数情况下更好。此外，它能够在30秒内完成所有数据集的处理。

VIII. 结论

我们设计了用于构建TMFG和DBHT的新型并行算法。我们表明，我们的算法通常能够生成优于完全连接和平均连接聚类的簇。我们相信，对于需要高效、准确地执行聚类的数据科学家而言，我们的实现将是实用工具箱中的有益补充。

致谢

本研究获得美国能源部早期职业奖#DE- SC0018947、美国国家科学基金会职业奖 #CCF-1845763、谷歌教师研究奖、谷歌研究学者奖、FinTech@CSAIL 倡议、美国国防高级研究计划局 SDH 奖 #HR0011- 18-3-0007，以及应用驱动架构（ADA）研究中心的支持。该中心是由半导体研究公司和美国国防高级研究计划局共同赞助的 JUMP 中心。

参考文献

[1] J. Ruan、A. K. Dean 和 W. Zhang，《BMC 系统生物学》，第 4 卷，第 1 期，第 8 页，2010年。[2] R. N. Mantegna，“金融市场中的层级结构”，《欧洲物理学杂志 B：凝聚态与复杂系统》，第 11 卷，第 1 期，第 193–197 页，1999年。[3] M. Tumminello、C. Coronnello、F. Lillo、S. Micciche 和 R. Mantegna，“基于相关性的网络中的生成树与自助法可靠性估计”。《国际分岔与混沌杂志》，第 17 卷，第 2319–2329 页，2007年 7 月。[4] M. Tumminello、T. Aste、T. Di Matteo 和 R. Mantegna，“一种过滤复杂系统信息的工具”，《美国国家科学院院刊》，第 102 卷，第 10 421–10 246 页，2005年 8 月。[5] G. P. Massara、T. di Matteo 和 T. Aste，“大数据网络过滤：三角剖分最大过滤图”，《复杂网络杂志》，第 5 卷，第 161–178 页，2017年。[6] N. Musmeci、T. Aste 和 T. Di Matteo，“金融市场结构与实体经济之间的关系：聚类方法比较”，《公共科学图书馆·综合》，第 10 卷，第 3 期，第 e0116201 页，2015年。[7] G.-J. Wang、C. Xie 和 S. Chen，“美国股市的多尺度相关网络分析：小波分析”，《经济互动与协调杂志》，第 12 卷，第 3 期，第 561–594 页，2017年。[8] P. T.-W. Yen 和 S. A. Cheong，“运用拓扑数据分析（TDA）和持久同调分析新加坡和台湾的股票市场”，《物理学前沿》，第 20 页，2021年。[9] W.-M. Song 和 B. Zhang，“多尺度嵌入式基因共表达网络分析”，《公共科学图书馆·计算生物学》，第 11 卷，第 11 期，第 e1004574 页，2015年。[10] M. J. Burton 等，“埃塞俄比亚和坦桑尼亚进行性瘢痕性沙眼的发病机制及其对疾病控制的启示：两项队列研究”，《公共科学图书馆·被忽视的热带病》，第 9 卷，0003763，2015 5，第 p. e .[11] L. Hubert 和 P. Arabie，“比较划分”，《分类学杂志》，第 2 卷，第 1 期，第 193–218 页，1985年。[12] J. Giffin，“设施布局的图论技术”，1984年。[13] W.-M. Song、T. Di Matteo 和 T. Aste，“通过拓扑嵌入图进行层级信息聚类”，《公共科学图书馆·综合》，第 7 卷，第 e31929 页，2012年 3 月。

[14] W.-M. Song、T. Di Matteo 和 T. Aste，“平面图中的嵌套层次结构”，《离散应用数学》，第 159 卷，2146，2011 17，第 2135 no. pp. – .
[15] J. Jaja，《并行算法导论》，1992年。[16] T. H. Cormen、C. E. Leiserson、R. L. Rivest 和 C. Stein，《算法导论》（第 4 版）。麻省理工学院出版社，2022年。[17] R. D. Blumofe 和 C. E. Leiserson，“通过工作窃取调度多线程计算”，《ACM学报》，第 46 卷，第 5 期，第 720–748 页，1999年 9 月。[18] U. Vishkin，“并行思维：一些基础的数据并行算法与技术”，2010年。[19] S. Rajasekaran 和 J. H. Reif，“最优和次对数时间随机并行排序算法”，《SIAM计算杂志》，第 18 卷，第 3 期，第 594–607 页，1989年。[20] J. Shun、G. E. Blelloch、J. T. Fineman 和 P. B. Gibbons，“通过优先级更新减少竞争”，载于 ACM算法与体系结构并行性研讨会，2013年，第 152–163 页。[21] D. M¨ ullner，“fastcluster：适用于 R 和 Python 的快速层次凝聚聚类例程”，《统计软件杂志》，第 53 卷，第 9 期，第 1–18 页，2013年。[22] S. Yu、Y. Wang、Y. Gu、L. Dhulipala 和 J. Shun，“ParChain：一种使用最近邻链进行并行层次凝聚聚类的框架”，《VLDB基金会会刊》，第 15 卷，第 2 期，第 285–298 页，2021年 10 月。[23] Y. Wang、S. Yu、Y. Gu 和 J. Shun，“欧几里得最小生成树与层次空间聚类的快速并行算法”，载于 SIGMOD，2021年，第 1982–1995 页。[24] L. Dhulipala、D. Eisenstat、J. Lacki、V. S. Mirrokni 和 J. Shi，“近线性时间的层次凝聚图聚类”，载于 ICML，2021年，第 2676–2686 页。[25] S. Rajasekaran，“高效并行层次聚类算法”，《IEEE并行与分布式系统汇刊》，第 16 卷，第 6 期，第 497–502 页，2005年。[26] C. F. Olson，“层次聚类的并行算法”，《并行计算》，第 21 卷，第 8 期，第 1313–1325 页，1995年。[27] W. Hendrix、M. M. A. Patwary、A. Agrawal、W.-k. Liao 和 A. Choud-hary，“共享内存平台上的并行层次聚类”，载于国际高性能计算会议，2012年，第 1–9 页。[28] E. Dahlhaus，“层次聚类的并行算法及其在分裂分解和奇偶图识别中的应用”，《算法杂志》，第 36 卷，第 2 期，第 205–240 页，2000年。[29] M. Dash、S. Petrutiu 和 P. Scheuermann，“高效并行层次聚类”，载于欧洲并行处理会议，2004年，第 363–371 页。[30] Q. Mao、W. Zheng、L. Wang、Y. Cai、V. Mai 和 Y. Sun，“面向大规模序列分析的线性对数时间并行层次聚类”，载于 IEEE国际数据挖掘会议，2015年，第 310–319 页。[31] R. Campello、D. Moulavi、A. Zimek 和 J. Sander，“用于数据聚类、可视化和异常值检测的层次密度估计”，TKDD，第 5:1–5:51 页，2015年。[32] V. Cohen-Addad、V. Kanade、F. Mallmann-Trenn 和 C. Mathieu，“层次聚类：目标函数与算法”，《ACM学报》，第 66 卷，第 4 期，第 1–42 页，2019年。[33] B. Wang、I. Rahal 和 A. Dong，“使用加权置信亲和度进行并行层次聚类”，《国际数据挖掘、建模与管理杂志》，第 3 卷，第 2 期，第 110–129 页，2011年。[34] S. Lattanzi、T. Lavastida、K. Lu 和 B. Moseley，“层次聚类方法并行化框架”，载于欧洲机器学习与数据库知识发现联合会议，2020年，第 73–89 页。[35] P. A. Kataki、M. R. Cappelle、L. R. Foulds 和 H. J. Longo，“最大权平面子图问题的新算法”，《第 52 届巴西运筹学研究研讨会论文集》，Jo˜ao Pessoa-PB，2020年。[36] I. H. Osman、B. Al-Ayoubi 和 M. Barake，“加权极大平面图问题的贪心随机自适应搜索过程”，《计算机与工业工程》，第 45 卷，第 4 期，第 635–651 页，2003年。[37] M. E. Dyer、L. R. Foulds 和 A. M. Frieze，“寻找最大权平面子图的启发式算法分析”，《欧洲运筹学杂志》，第 20 卷，第 1 期，第 102–114 页，1985年。
[38] R. Cimikowski，“最大平面子图问题若干启发式算法的分析”，载于ACM-SIAM离散算法研讨会，1995年，第322–331页。[39] P. Eades、L. Foulds和J. Giffin，“识别最大权平面子图的一种高效启发式算法”，载于组合数学IX，1982年，第239–251页。[40] Z. Ma、Z. Kang、G. Luo、L. Tian和W. Chen，“迈向聚类友好型表示：通过图滤波进行子空间聚类”，载于ACM MM，2020年，第3081–3089页。[41] N. Tremblay、G. Puy、P. Borgnat、R. Gribonval和P. Vandergheynst，“利用随机信号的图滤波加速谱聚类”，载于ICASSP，2016年，第4094–4098页。[42] T. Jebara和V. Shchogolev，“用于谱聚类的B-匹配”，载于欧洲机器学习会议，2006年，第679–686页。[43] T. Jebara、J. Wang和S.-F. Chang，“用于半监督学习的图构建和b-匹配”，载于机器学习国际会议论文集，2009年，第441–448页。[44] Y. Emek、S. Kutten、M. Shalom和S. Zaks，“分层b-匹配”，载于信息学理论与实践当前趋势国际会议，2021年，第189–202页。[45] T. Aste，“DBHT”， 2022, [MATLAB Central 文件交换]https://www.mathworks.com/matlabcentral/fileexchange/46750-dbht。[46] G. E. Blelloch和B. M. Maggs，“并行算法”，载于计算机科学与工程手册，1997年，第277–315页。[47] U. Meyer和P. Sanders，“Δ-步进：一种可并行化的最短路径算法”，《算法杂志》，第49卷，第1期，第114–152页，2003年。[48] X. Dong、Y. Gu、Y. Sun和Y. Zhang，“用于并行最短路径的高效步进算法及实现”，载于ACM算法与体系结构并行性研讨会（SPAA），2021年，第184–197页。[49] L. Dhulipala、G. Blelloch和J. Shun，“Julienne：一个使用工作高效分桶的并行图算法框架”，载于ACM算法与体系结构并行性研讨会（SPAA），2017年。[50] P. N. Klein和S. Subramanian，“一种用于单源最短路径的随机化并行算法”，《算法杂志》，第25卷，第2期，1997年11月。[51] E. Cohen，“用于无向最短路径的多对数时间和近线性工作量近似方案”，《ACM学报》，第47卷，第1期，2000年1月。[52] B. Sinha、B. Bhattacharya、S. Ghose和P. Srimani，“一种用于计算图的最短路径和直径的并行算法及其VLSI实现”，IEEE计算机汇刊，第C-35卷，第11期，1986年11月。[53] R. Seidel，“关于全点对最短路径问题”，载于ACM计算理论研讨会，1992年。[54] M. Papaefthymiou和J. Rodrigue，“并行最短路径算法的实现”，载于DIMACS离散数学与理论计算机科学丛书，1994年。[55] K. Madduri、D. A. Bader、J. W. Berry和J. R. Crobak，“用于求解大规模图实例的并行最短路径算法实验研究”，载于算法工程与实验（ALENEX），2007年，第23–35页。[56] A. Karczmarz和P. Sankowski，“一种确定性并行全点对最短路径算法及其应用”，载于ACM-SIAM离散算法研讨会（SODA），第255–272页。[57] M. R. Henzinger、P. Klein、S. Rao和S. Subramanian，“用于平面图的更快最短路径算法”，计算机与系统科学杂志，第55卷，第1期，第3–23页，1997年。[58] G. E. Pantziou、P. G. Spirakis和C. D. Zaroliagis，“平面图中最短路径的高效并行算法”，载于斯堪的纳维亚算法理论研讨会，1990年，第288–300页。[59] G. N. Federickson，“平面图最短路径的快速算法及其应用”，SIAM计算杂志，第16卷，第6期，第1004–1022页，1987年。[60] J. L. Tr¨aff和C. D. Zaroliagis，“一种用于平面有向图单源最短路径问题的简单并行算法”，载于非规则结构问题并行算法国际研讨会，1996年，第183–194页。[61] P. Klein和S. Subramanian，“一种用于平面图最短路径的线性处理器多对数时间算法”，载于IEEE计算机科学基础研讨会，1993年，第259–270页。[62] D. Z. Chen和J. Xu，“平面图中的最短路径查询”，载于ACM计算理论研讨会，2000年，第469–478页。[63] H. N. Djidjev、G. E. Pantziou和C. D. Zaroliagis，“计算平面图中的最短路径与距离”，载于自动机、语言与编程国际研讨会，1991年，第327–338页。

[64] H. N. Djidjev，“平面有向图中最短路径查询的高效算法”，载于国际图论计算机科学概念研讨会，1996年，第151–165页。[65] J. Fakcharoenphol和S. Rao，“平面图、负权边、最短路径与近线性时间”，计算机与系统科学杂志，第72卷，第5期，第868–889页，2006年。[66] P. Klein、S. Rao、M. Rauch和S. Subramanian，“用于平面图的更快最短路径算法”，载于*ACM*计算理论研讨会，1994年，第27–37页。[67] P. N. Klein、S. Mozes和O. Weimann，“具有负长度的有向平面图中的最短路径：一种线性空间 $O(n \log^2 n)$时间算法”，*ACM*算法汇刊，第6卷，第2期，2010年4月。[68] S. Mozes和C. Wulff-Nilsen，“具有实数长度的平面图中的最短路径，可在 $O(n\log^2 n/\log\log n)$时间内完成”，载于欧洲算法研讨会，2010年，第206–217页。[69] P. Gawrychowski、S. Mozes、O. Weimann和C. Wulff-Nilsen，“平面图精确距离预言机的更优权衡”，载于*ACM-SIAM*离散算法研讨会（*SODA*），第515–529页。[70] V. Pan和J. Reif，“路径代数问题的快速高效求解”，计算机与系统科学杂志，第38卷，510，1989年3，第494页。–。[71] G. E. Blelloch、D. Anderson和L. Dhulipala，“ParlayLib——一个用于共享内存多核机器上并行算法的工具包”，载于*ACM*算法与体系结构并行性研讨会（*SPAA*），2020年，第507–509页。[72] B. Bahmani、B. Moseley、A. Vattani、R. Kumar和S. Vassilvitskii，“可扩展k-means++”，《*VLDB*基金会会刊》，第5卷，第7期，第622–633页，2012年3月。[73] G. Thompson，“mpi-scalablekmeanspp”，2018年，https://github.com/gzt/mpi-scalablekmeanspp。[74] M. Luci´nska和S. T. Wierzcho´n，“基于k近邻图的谱聚类”，载于*IFIP CISIM*，2012年，第254–265页。[75] G. Guennebaud、B. Jacob 等人，“Eigen”，2010年，http://eigen.tuxfamily. org。[76] N. X. Vinh、J. Epps和J. Bailey，“用于聚类比较的信息论度量：变体、性质、归一化及随机校正”，《机器学习研究杂志》，第11卷，第2837–2854页，2010年12月。[77] H. A. Dau 等人，“UCR 时间序列分类档案”，2018年10月，https://www.cs.ucr.edu/~eamonn/time series data 2018/。[78] G. Marti、F. Nielsen、M. Bi´nkowski和P. Donnat，“对金融领域二十年相关性、层次结构、网络和聚类的综述”

– – –

市场，”《信息几何进展》，第 245–274 页，2021年。[79] V. Tola、F. Lillo、M. Gallegati 和 R. N. Mantegna，“用于投资组合优化的聚类分析”，《经济动态与控制杂志》，第 32 卷，第 1 期，第 235–258 页，2008 年。



图 12： 数据点之间的相关矩阵

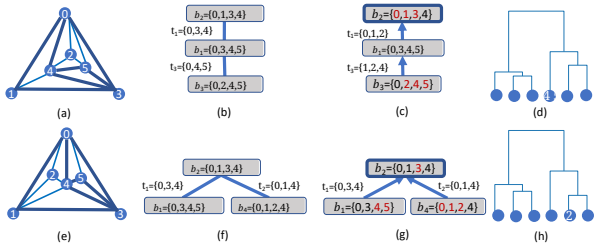


图 13： (a)–(d)：使用前缀=1的示例。(e)–(h)：使用前缀=3的示例。

附录

现在，我们给出一个更详细的示例，说明使用较大的前缀可以获得更好的聚类结果。此示例中数据点之间的相关性如图 12 所示，真实聚类由两个簇$\{0,1,2\}$ 和 $\{3,4,5\}$组成。我们将看到，使用 PREFIX=3 生成的树状图能够恢复这一真实聚类，而使用 PREFIX=1 生成的树状图则不能。请看图 13 中的示例。图 13(a) 是使用 PREFIX=1 的 TMFG，图 13(b) 是使用 PREFIX=3的 TMFG。对于这两种前缀大小，TMFG 算法都从 4-团 $\{0,1,3,4\}$开始。当 PREFIX=1时，我们先将顶点 5 添加到$t_1 = \{0,3,4\}$，然后将顶点 2 添加到 $t_3 = \{0,4,5\}$。另一方面，当 PREFIX=3时，我们在同一轮中添加顶点 2 和 5。由于添加顶点 2 时不存在 t_3，因此我们将顶点 2 添加到 $t_2 = \{0,1,4\}$。在图 13(c) 和 (g) 中，顶点被分配到其标记为红色的气泡中。

当 PREFIX=1 时，我们得到图 13(d) 所示的树状图。我们可以看到，无法恢复真实聚类，因为顶点 2 被放在了 t_3 而不是 t_2，这是因为 $corr(2,5) = 0.42$ 略大于 $corr(2,1) = 0.41$，但这种差异可能是由数据集中的噪声造成的。当 PREFIX=3 时，我们得到图 13(h) 所示的树状图，并且可以通过过滤掉噪声来获得真实聚类结果。