高技术通讯 2012 年 第22 卷 第11 期:1169 ~ 1175

doi:10.3772/j.issn.1002-0470.2012.11.010

层次凝聚聚类算法的动态分析与准则函数设计①

王 洋②*** 涂登彪*** 安明远* 孙凝晖** 王伟平**

(*中国科学院研究生院 北京 100190)
(**中国科学院计算机系统结构重点实验室 北京 100190)
(***国家计算机网络应急技术处理协调中心 北京 100029)

摘 要 为提高层次凝聚聚类(HAC)算法的执行效率和结果质量,对其进行了动态分析,研究了一次合并对后续合并的影响。分析表明,合并两个类会生成一个新类,并使被合并的类的共享邻居的邻居数减小1;当新生成的类或邻居数减小的类参与后续合并时,会影响执行效率;一次合并会改变参与合并的类和它们的候选邻居之间的准则函数值,从而影响后续合并提高质量的程度。基于上述分析并结合模块性的定义,研究了现有准则函数对凝聚过程的影响以及它们的缺陷,并设计了两个新的准则函数。在大量数据集上的实验表明,新的准则函数提高了层次凝聚聚类算法的执行效率和结果质量。

关键词 层次凝聚聚类(HAC)算法,准则函数,模块性,聚类分析

0 引 言

层次凝聚聚类(hierarchical agglomerative clustering algorithm, HAC)算法[1]是聚类分析的典型算法。聚类分析有广泛应用。它是根据数据对象特性对数据对象进行分类的方法,其分类原则是属于同一个类的数据对象之间极其相似,而属于不同类的数据对象之间极不相似。HAC 算法的主要过程是:首先把每个数据对象看作一个类,并定义一个以两个类为参数的准则函数;然后计算每两个相邻的类对应的准则函数值;最后,迭代地选取准则函数值最高的两个类合并成一个新类,并计算这个新类和其邻居对应的准则函数值,直到达到终止条件。只要能够定义准则函数,HAC 算法即可工作,因此它可以处理文档[2]、分类数据[3]、欧几里德空间的点[4]等各种类型的数据,广泛应用于模式识别、图像处理等领域。由于 HAC 算法使用图刻画数据对象之间的关系,因此被自然地应用于复杂网络的聚类分析[5-9]。复杂网络上的聚类结果称为社区结构,它可以用来理解网络的功能以及节点间关系。本文以复杂网络的社区结构分析为背景研究 HAC 算法。

使用 HAC 算法首要的问题是定义准则函数,因为它决定了整个凝聚过程,包括执行效率和最终聚类结果的质量。文献[5-7]基于模块性(modularity)[10]定义了不同的准则函数,实验发现,这些准则函数有的产生低质量的结果,有的使 HAC 算法过于耗时,都不能有效地处理大规模复杂网络。为了设计更加有效的准则函数,提高 HAC 算法的执行效率和聚类结果的质量,本研究在文献[11]的基础上对 HAC 算法进行了动态分析,同时也分析了已有的准则函数的缺陷,并设计了两个能显著提高 HAC 算法的执行效率和质量的新的准则函数,而且在多个数据集上进行了实验,验证了对 HAC 算法及已有准则函数的分析结果,以及新的准则函数的有效性。通过对实验结果的分析,也指出了使用低质量聚类算法分析聚类结果规模分布的局限性。

1 HAC 算法效率分析

HAC 算法本质上是一个迭代的过程,对于确定的输入,凝聚过程中每一次对类的选择都是基于上一次合并后重新计算的准则函数值,而且每个参与合并的类,也是由以前的合并生成的,因此一次合并会对后续合并产生影响,进而影响 HAC 算法的执行

① 863 计划(2009AA01A129)和国家自然科学基金(60903047)资助项目。
② 男,1981 年生,博士生;研究方向:数据挖掘;联系人,E-mail: aaron@ncic.ac.cn
(收稿日期:2011-04-25)

效率和聚类结果的质量。如果一次合并能够生成邻居较少的类，并且使多个类的邻居减少，则能够提高后续合并的效率，因此提升 HAC 算法效率的关键在于控制被合并的类的邻居数和共享邻居数。另外，合并两个类会产生新的类，它们的邻居与新类的准则函数值被重新计算，这会影响它们能否在以后的凝聚过程中被选择合并，或者影响它们被合并时提高质量的程度。

HAC 算法一般使用两个数据结构。第一个数据结构是为每一个类分配的，称为局部数据结构，它为每个邻居保存一个 <类 ID，准则函数值，其他信息 > 形式的元组。文献[5-8]都对局部数据结构进行了研究，为了便于讨论，假定局部数据结构中的元组使用链表组织，通过平衡二叉树索引每个邻居，并使用堆将邻居根据准则函数值排序。第二个数据结构是建立在所有类上的一个大顶堆[12]，记作 H。在初始化阶段，每个类 c 从自己的邻居集合 $S_n(c)$ 中找到候选邻居 $C_n(c)$，使得 $C_n(c)$ 满足对任意 $c' \in S_n(c)$，有 $f(c, C_n(c))$ 不小于 $f(c, c')$。每个类找到候选邻居后，通过调整全局堆 H 对所有类进行排序，排序的规则是：如果 $f(c, C_n(c))$ 小于 $f(c', C_n(c'))$，则 c 小于 c'。当 H 调整结束后，HAC 算法初始化过程结束，位于 H 顶端的类和它的候选邻居就是准则函数值最大的两个类。

初始化完成后，HAC 算法进入凝聚过程。凝聚过程由一系列合并组成，这是 HAC 算法最耗时的部分。本节首先分析合并的效率，然后根据每次合并对后续合并的影响分析整个凝聚过程的效率。

1.1 合并两个类的效率

一次合并的详细过程如算法 1 所示，其中 c_i 和 c_j 是被合并的类，它们和它们的邻居都是参与合并的类。由于局部数据结构中的平衡二叉树使每个类的邻居集合变得有序，算法 1 第(3)步可以采用归并的方法获得 c_i 的邻居集合，花费的时间为 $O(|S_n(c_i)| + |S_n(c_j)|)$。算法 1 第(4)步中 c_j 的每个邻居 (c_k) 花费的时间为 $O(\log S_n(c_k))$。一般计算准则函数值花费 $O(1)$ 时间，因此算法 1 第(5)步花费的时间为 $O(|S_n(c_i) \cup S_n(c_j)|)$。如果邻居变化较多，类 c_i 在算法 1 第(6)步重建局部数据结构中的平衡二叉树和堆，花费的时间为 $O(|S_n(c_i) \cup S_n(c_j)| \log |S_n(c_i) \cup S_n(c_j)|)$，而 c_i 的每个邻居 (c_k) 花费的时间为 $O(\log |S_n(c_k)|)$。在算法 1 第(7)步，类 c_i 和它的每个邻居都可能导致调整 H，共花费时间 $O(|S_n(c_i) \cup S_n(c_j)| \log n)$，其中 n 是原始图的节点数。由于 n 大于 $|S_n(c_i) \cup S_n(c_j)|$，因此一次合并的时间复杂度为 $O(|S_n(c_i) \cup S_n(c_j)| \log n)$。

算法 1：两个类的合并

(1) 取 H 的顶端元素 c_i；
(2) c_j 为 c_i 的候选邻居；
(3) c_i 将 c_j 的邻居并入自己的邻居集合；
(4) 每个 c_j 的邻居更新自己的邻居集合，把 c_j 替换为 c_i；删除 c_j；
(5) 计算 c_i 的和它的邻居之间的 f 值；
(6) 新的 c_i 和它的每个邻居都重新寻找自己的候选邻居；
(7) 根据重新寻找候选邻居的结果调整 H；

从复杂度来看，合并的效率取决于参与合并的类的数量，但是并不是每个参与合并的类都一定导致调整 H，而且一次调整 H 花的时间也往往少于 $O(\log n)$，因此合并过程中其他步骤的代价也决定执行效率[11]。整个合并过程中，第(3)步、第(5)步和第(7)步的代价取决于被合并的类的邻居数，第(4)步和第(6)步的代价取决于每个参与合并的类的邻居数。因此，提高合并两个类的执行效率的途径是：(1)减小被合并的类的邻居数；(2)减小每个参与合并的类的邻居数。

1.2 凝聚过程的效率

HAC 算法凝聚过程的执行代价是所有合并的代价的总和，因此提高效率的方法是降低每次合并的代价。对一次合并来说，被合并的类或者其他参与合并的类可能是直接根据输入数据初始化得到的，也可能是在以前的合并中产生的，还可能是被以前的合并影响过的。不考虑输入数据的因素，一次合并的代价是由之前的合并决定的。下边以合并类 c_i 和类 c_j 为例(如图 1)分析一次合并如何影响后续的合并。

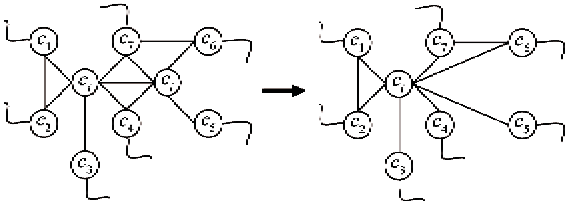

图 1 两个类的合并

一方面，c_j 被合并到 c_i 中会产生一个新的 c_i，其邻居集合为 $S_n(c_i) \cup S_n(c_j) - \{c_i\} - \{c_j\}$。图 1 中类 c_i 合并前有 6 个邻居，而合并后有 7 个邻居。另

一方面,如果一个类是被合并的两个类的共享邻居(例如图1的c_7),那么合并后这个类的邻居数量会减小1。一般来说,如果两个类有较少的邻居,那么合并后一般会产生一个有较少邻居的新类,而如果两个类有较多的共享邻居,那么合并后使得较多的类的邻居数减小。在合并完成后,邻居数改变的类(如c_i和c_7等)会参与到以后的合并中。在上一节已经分析,减小被合并的类的邻居数,或者减小参与合并的类的邻居数,能够提高合并的执行效率,因此合并邻居较少的两个类,或者合并共享邻居较多的两个类,会对后续的合并产生有利影响,这样,合并迭代地进行,就能够提高整个凝聚过程的执行效率。

为了更好地分析HAC算法的效率,图2给出了同一个图上4个不同凝聚过程的例子。原始图有6个节点,构成初始的6个类。为了便于比较,每次合并的代价简化为$O(|S_n(c_i)\cup S_n(c_j)|\log 6)$,即正比于参与凝聚的类的个数。例a中第1次被合并的类是c_1和c_2,其他参与合并的类是c_3和c_5,因此代价为4log6。4个凝聚过程的执行代价分别是16log6、18log6、20log6和18log6。

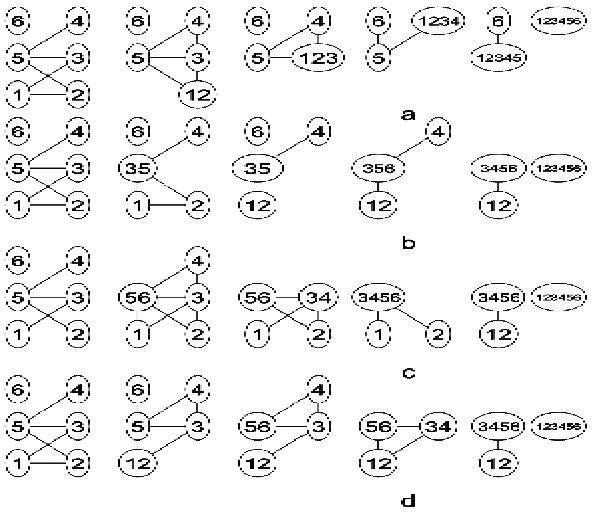

图2　不同凝聚过程示例

例a的特点是,它总是合并邻居较少的类。类c_5有5个邻居,是邻居最多的类,但是c_5的邻居在第1、2、3次合并中减少了,当第4步中合并c_5时,它只有2个邻居,可见前边的合并会对后续的合并产生决定性的影响。在例b中,首先合并了类c_3和c_5。虽然c_3和c_5共享了很多邻居,但是由于它们总的邻居很多,最终生成的c_{35}有大量的邻居,这导致在后续的合并中每次与c_{35}合并的代价都很高。比较例a和例b可知合并邻居较少的类可以提高后续合并的效率。例d和例c的唯一区别是,例d中首先合并了c_1和c_2,而例c中在第4步合并。由于c_1和c_2共享邻居c_3和c_5,合并c_1和c_2能够减小c_3和c_5的邻居数,从而降低后续合并的代价。虽然就合并c_1和c_2这一步来说,例d的代价是4log6,而例c的代价是3log6,但是由于c_3和c_5在后续合并中不断降低代价,例d比例c总的执行代价更低,可见合并共享邻居较多的类可以提高后续合并的效率。

2　基于模块性的准则函数

本节首先介绍了模块性的定义,分析了一次合并对后续合并提升质量程度的影响,然后分析了现有准则函数存在的问题。在此基础上,提出了新的准则函数,有效地控制了凝聚过程。

2.1　模块性与聚类结果质量

模块性是广泛使用的量化复杂网络社区结构强度的指标,被用来评价复杂网络上聚类结果的质量[10]。给定图G的一个划分$P\{c_1,c_2,\cdots,c_n\}$,它的模块性定义如式

$$Q(P)=\sum_{c_i\in P}Q(c_i)=\sum_{c_i\in P}\left(\frac{l_{ii}}{L}-\left(\frac{d(c_i)}{2L}\right)^2\right) \tag{1}$$

所示,其中L为图G的边数,l_{ij}表示类c_i和类c_j的成员(G的节点)之间的边数,$d(c_i)$表示类c_i的成员的度的总和。在模块性的定义中,l_{ii}/L意味着类c_i的成员之间相连的边数,而$(d(c_i)/(2L))^2$意味着G的所有边按照随机分布落在类c_i内的边数,因此一个划分对应的模块性越高,说明每个类内的节点间联系越紧密。对应到聚类的概念上,模块性高说明类内节点间非常相似,聚类质量好。

合并两个类c_i和c_j对应的模块性变化ΔQ如式

$$\begin{aligned}\Delta Q(c_i,c_j)&=Q(c_i\cup c_j)-Q(c_i)-Q(c_j)\\&=\frac{1}{L}\left(l_{ij}-\frac{d(c_i)d(c_j)}{2L}\right)\end{aligned} \tag{2}$$

所示。ΔQ值大于0意味着两个类是联系紧密的,合并它们可以提高聚类结果的质量。已有的基于模块性的准则函数都以ΔQ作为因子,HAC算法通过不断合并ΔQ值大于0的两个类来完成聚类,提高聚类结果质量的方法是提高每次合并的两个类的ΔQ值。需要指出的是,一次合并会影响后续合并的ΔQ值。类c_i和类c_j合并后,$d(c_i)$会增加。如果类c_k是c_i的邻居而不是c_j的邻居,由于l_{ik}不变而$d(c_i)$增加,则$\Delta Q(c_i,c_k)$会变小;如果c_k是c_i和c_j的共

享邻居，则 l_{ik} 显著增加，如果 $d(c_i)$ 变化较小，则 $\Delta Q(c_i, c_k)$ 会变大。如果 c_i 是 c_k 的候选邻居，则 $\Delta Q(c_i, c_k)$ 的变化会影响 c_k 被合并时的 ΔQ 值，从而影响聚类结果的质量。

2.2 现有准则函数存在的问题及分析

在文献[5]中，ΔQ 被直接用作准则函数（f_{dQ}），这样，HAC 算法每次合并 ΔQ 值最大的两个类。f_{dQ} 的一个缺陷是它会导致超大规模的类，这一现象可以通过分析一次合并对后续合并的影响来解释。首先，两个类对应的 f_{dQ} 值主要由两个类之间的边数决定，因为在式(2)中 $d(c_i)d(c_j)$ 一般远远小于 $2L$。例如，一个图有 10000 条边和 1000 个节点，两个度均为 10 的相邻节点 v_1 和 v_2 被初始化为类 c_1 和 c_2，那么 l_{12} 等于 1，而 $d(c_i)d(c_j)/(2L)$ 等于 5×10^{-3}。其次，两个类合并后，新生成的类可能和某个邻居之间有更多的边。假设类 c_k 是 c_i 和 c_j 的共享邻居，则新生成的类 c_i 和 c_k 之间的边数会变为 $l_{jk}+l_{ik}$。在算法开始阶段，两个类之间的边数都为 1，因此如果两个有共享邻居的类合并后，新类 c_i 和共享邻居 c_k 之间的边数变为 2。由于 f_{dQ} 的值由两个类之间的边数决定，因此 $f_{dQ}(c_i, c_k)$ 会变为全局最大，导致 c_i 和 c_k 合并，而这一合并会导致 c_i 和 c_k 的某个共享邻居与新的 c_i 之间的 f_{dQ} 又成为全局最大，因此类 c_i 会继续和邻居合并。一个类在几次合并后会有更多成员，和更多的类有共享邻居，因此会与多个邻居以较多的边相连，对应的 f_{dQ} 显著高于其他的类，这使得一个较大的类不断地吞并邻居，直到该类的规模足够大，以至于它和邻居之间的 f_{dQ} 值不再是整个图中最大的。如果图上节点间共享邻居的情况普遍存在而且图足够大，则形成超大类的过程会重复多次。超大规模的类往往有大量的邻居，因此在它们与某个邻居合并过程中会有大量的类参与，这是以 f_{dQ} 为准则函数导致 HAC 算法低效率的根本原因。

很多类以超大规模的类为候选邻居，在超大规模的类形成过程中，这些类与候选邻居之间的 ΔQ 值不断降低，因此在后续合并时提升的质量不多。一些类仅以超大规模的类为邻居，这些类的规模较小，与超大规模的类之间的边数也很少，因此一直没有被合并，这是以 f_{dQ} 为准则函数在产生超大规模的类同时产生大量很小规模的类的原因。

为了避免较大的类不断吞并较小的邻居，文献[6]引入平衡性的概念，提出了准则函数 f_m，如式

$$f_m(c_i,c_j)=\Delta Q(c_i,c_j)\frac{\min\{|c_i|,|c_j|\}}{\max\{|c_i|,|c_j|\}} \quad (3)$$

所示。相比于 f_{dQ}，f_m 确实能够控制合并的类的邻居数，提高 HAC 算法的执行效率。f_m 的一个缺陷是它会在凝聚过程的后期选择两个成员较多的类合并，因为这样的类有较高的平衡性，而成员较多的类的邻居也较多，因此合并的执行效率较低。

文献[7]中给出了准则函数 f_a，如式

$$f_a(c_i,c_j)=\frac{\Delta Q(c_i,c_j)}{\min\{d(c_i),d(c_j)\}} \quad (4)$$

所示。一般来说，如果类 c_i 对应的 $d(c_i)$ 较小，则 c_i 的邻居也比较少，因此 f_a 在一定程度上控制了合并的类的邻居数。但是，f_a 只强调合并的两个类中的一个，不能有效地避免邻居很多的类与邻居很少的类合并的情况，因此对 HAC 算法效率的提高是有限的。

根据式(4)，一个类 (c_i) 倾向于选择成员的度的和很小的类 (c_j) 作为候选邻居，因此两个类合并后 $d(c_i)$ 的变化很小。如果类 c_k 以 c_i 为邻居但不以 c_j 为邻居，则合并后 $\Delta Q(c_i, c_k)$ 下降很小，这样即使 c_i 作为 c_k 的候选邻居，c_k 在以后被合并时 ΔQ 值下降也很小。如果 c_k 是 c_i 和 c_j 的共享邻居，则 l_{ik} 显著增加，由于 $d(c_i)$ 变化很小，则 $\Delta Q(c_i, c_k)$ 会变大，这有利于 c_k 合并时提高质量。

2.3 新的准则函数

本文给出 f_{n*} 和 f_{m*} 两个新的准则函数：

$$f_{n*}(c_i,c_j)=\frac{\Delta Q(c_i,c_j)}{|S_n(c_i)|\times|S_n(c_j)|} \quad (5)$$

$$f_{m*}(c_i,c_j)=\Delta Q(c_i,c_j)\frac{|S_n(c_i)\cap S_n(c_j)|+2}{|S_n(c_i)|\times|S_n(c_j)|} \quad (6)$$

f_{n*} 倾向于选择邻居较少的类进行合并，这样在合并后产生的新类的邻居也较少。随着凝聚过程的不断进行，邻居较少的类不断被产生，每次参与合并的类的邻居都较少，因此整个凝聚过程有更高的执行效率。f_{m*} 在 f_{n*} 的基础上强调了两个类的共享邻居数，这样可以使更多的类在合并后邻居减少，进一步提高执行效率。在式(6)分子中加了常数 2 是因为 $|S_n(c_i)\cap S_n(c_j)|$ 可能导致两个类的准则函数值为 0，而令 $N(c_i)=S_n(c_i)\cup\{c_i\}$，则有 $|N(c_i)\cap N(c_j)|=|S_n(c_i)\cap S_n(c_j)|+2$。计算两个类的邻居集合的交集是耗时的，而准则函数又被频繁地计算，这会在 HAC 算法执行过程中耗费过多的时间。

邻居较少的类一般其成员的度的和也较小，因

此f_{n*}和f_{m*}都能够避免生成很大的类,这在一定程度上避免了聚类结果质量的降低。f_{m*}会选择共享邻居较多的两个类合并,这意味着合并后新类与多个邻居之间的边数增加,因此新类与这些邻居的ΔQ值增加。这使得后续合并中能够合并ΔQ值更大的类,最终提高聚类结果的质量。

3　实验结果及分析

本节测评每个准则函数对HAC算法的执行效率和聚类结果的质量的影响。HAC算法使用C++语言实现,可执行程序运行在RHEL as4系统上,终止条件设为ΔQ不大于0。实验使用的计算机采用AMD Opteron-2216处理器,有4GB内存。实验中使用的数据集是复杂网络研究人员整理并公开的[13]。

3.1　执行效率

表1给出了HAC算法使用不同准则函数的执行效率。f_{n*}在所有的数据集上对应的执行效率都显著超越了已有的准则函数,这表明每次合并时两个类的邻居数是影响整个凝聚效率的关键因素。在规模最大的数据集ndedu上,f_{n*}比f_{dQ}快100多倍,比已有准则函数中最快者(f_a)仍然快7倍。

表1　f_{dQ}对应执行时间及其他准则函数的加速比

数据集	f_{dQ}(s)	$Speedup = T(f_{dQ})/T(f)$			
		f_m	f_a	f_{n*}	f_{m*}
power	0.1	0.71	0.97	1.05	0.83
hep	0.8	3.20	3.95	4.52	2.59
pgp	1.7	2.13	4.24	5.55	2.20
cm99	5.8	8.01	7.88	10.68	5.40
aph	30.8	17.61	5.33	20.29	4.66
as	26.6	1.05	4.26	11.42	3.42
cm03	66.4	31.49	15.95	43.52	19.14
cm05	156.5	47.43	11.72	61.78	27.33
ndedu	4551.7	3.20	12.52	101.08	36.19

通过表1还可以看出,数据集越大,f_{n*}的优势越明显。根据第1.2节的分析,选择邻居较少的类进行合并会生成邻居较少的类,提高后续合并的效率,数据集越大,f_{n*}提高效率的机会越多,因此最终效果越明显。由于计算f_{m*}需要计算两个类的邻居集合的交集,耗费大量的时间,因此使用f_{m*}并不比f_{n*}更快,如表1。但是合并共享邻居较多的类确实能够使更多的类邻居减少,从而降低寻找候选邻居的代价[11],提高执行效率。

图3展示了cm03数据集上使用不同准则函数时合并的类的邻居数,图上每个点代表1000次合并(即一个"阶段")的类的邻居数的平均值。使用f_{dQ}在两个阶段后就产生了超大规模的类,这个类最多时有5724个邻居。cm03数据集中节点的度最大值为202,可见超大规模的类是不断合并邻居形成的。到第16阶段为止,f_{dQ}对应的凝聚过程生成了4个超大规模的类,这些类有大量邻居,导致了凝聚过程的低效率。使用f_a在前期的凝聚过程中避免了超大规模的类的产生,但是在后期仍然有三个阶段形成了邻居很多的类。使用f_m在整个凝聚过程中都能够较好地控制类的邻居数,但是在第25、26阶段合并的类的邻居数波动较大。与f_m相比,f_{n*}和f_{m*}对邻居数的控制更有效,虽然f_{n*}的优势在前期并不明显,但是随着合并的进行优势逐渐增加,可见前期的合并对后期的合并产生决定性的影响。

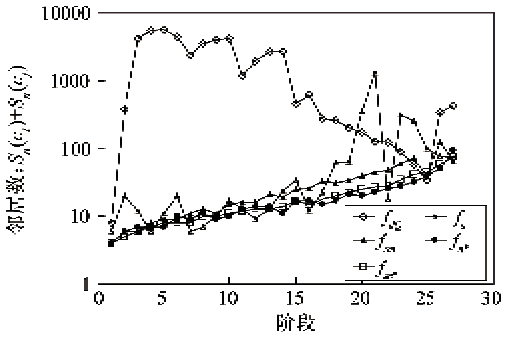

图3　cm03上合并的类的邻居数统计

3.2　聚类结果的质量

本文用模块性来评价聚类结果的质量。获得最高质量的结果,即寻找一个图的最高模块性的划分,是一个NP完全问题[14],而且对任一网络使用任何算法,其聚类结果对应的模块性都不会超过1。在所有数据集上,f_{dQ}对应的聚类结果质量(如表2)都超过了0.6,甚至在一些数据集上超过了0.9,因此进一步提升聚类结果的质量并不容易。从表2可以看出,f_{n*}在所有数据集上对应的质量都好于f_{dQ},并且在一些数据集上优势明显。f_a、f_{n*}和f_{m*}在质量上差别较小,平均来说,f_a对应的结果质量超过f_{n*},而f_{m*}又超过了f_a,可见合并共享邻居较多的类能够有效改善聚类结果的质量。除as数据集之外,f_m在所有数据集上对应的聚类结果质量都比f_{dQ}低,而且有的非常严重(表中用下划线标出),这说明在合并类时强调类之间的平衡性是不合理的。

表 2 聚类结果质量比较

数据集	模块性				
	f_{dQ}	f_m	f_{n*}	f_{m*}	f_c
power	0.933	0.927	0.935	0.936	0.935
hep	0.784	0.781	0.807	0.815	0.817
pgp	0.855	0.830	0.864	0.876	0.875
cm99	0.761	0.759	0.817	0.826	0.823
aph	0.621	0.546	0.678	0.704	0.673
as	0.627	0.628	0.643	0.631	0.652
cm03	0.673	0.620	0.730	0.740	0.736
cm05	0.626	0.553	0.689	0.703	0.696
ndedu	0.927	0.916	0.935	0.935	0.941

根据第 2 节的分析，一次合并会影响其它的类与候选邻居之间的 ΔQ 值。图 4 给出了 cm03 数据集上每个阶段提升的模块性的比较，图上每个点表示到该阶段为止所有合并提升的模块性的总和。使用 f_{dQ} 作为准则函数会每次合并都选择模块性增加最大的两个类，这实际上使 HAC 算法变成了一个贪心优化的算法。在前 25 个阶段中，f_{dQ} 对应的凝聚过程提升的模块性比其他准则函数都高，但是在最后两个阶段中模块性几乎无变化，可见在超大规模的类生成过程中大量的类和它们的候选邻居之间的 ΔQ 值已被严重降低。f_{c*}、f_n 和 f_{m*} 在前期提升模块性较慢，而后期提升较快，可见前期合并较小的类增加了后期的提升模块性的机会。

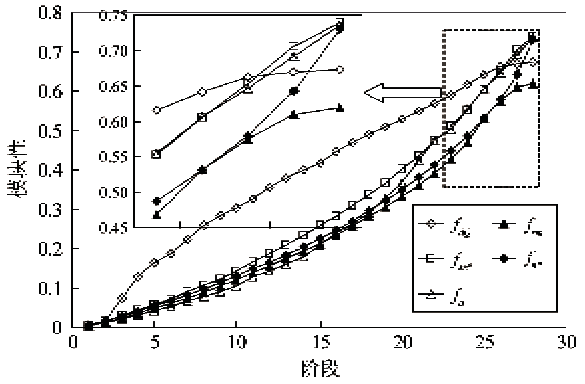

图 4 cm03 上质量提升过程比较

3.3 类的规模分布

复杂网络的社区规模分布（即最终聚类结果的规模分布）体现了网络的结构特征。文献[15]使用 HAC 算法以 f_{dQ} 为准则函数分析复杂网络后发现聚类结果中类的规模是幂律（Power-law）分布的，即规模为 k 的类的个数与 k 的对数成正比，这与本文的实验结果吻合。在 cm03 上，使用 f_{dQ} 最终产生 331 个类，并且规模服从幂律分布，其中最大的 4 个类包含了超过整个图 2 和图 3 的节点，而有 208 个类的规模不足 100。根据第 2.2 节的分析，使用 f_{dQ} 会导致少数超大规模的类和大量很小的类，这是导致聚类结果中类的规模呈幂律分布这一现象的主要原因。

图 5 给出了使用其他 4 个准则函数的聚类结果的规模分布，类按照规模从大到小编号，其中 f_m 对应的曲线有明显的断裂（gap），在文献[16]中使用基于 Betweenness 的层次分裂聚类算法研究聚类结果的规模分布时也发现了这一现象。需要指出的是，文献[16]所用的聚类算法，和以 f_m 为准则函数的 HAC 算法对应的聚类结果质量都很低，而从图 5 中 f_c、f_{n*} 和 f_{m*} 对应的曲线可以看出，在高质量的聚类结果中类的规模分布曲线没有断裂现象。根据上述分析可知，聚类结果的幂律分布以及规模分布曲线的断裂都是特定算法所产生的结果，不体现网络结构的本质特征。

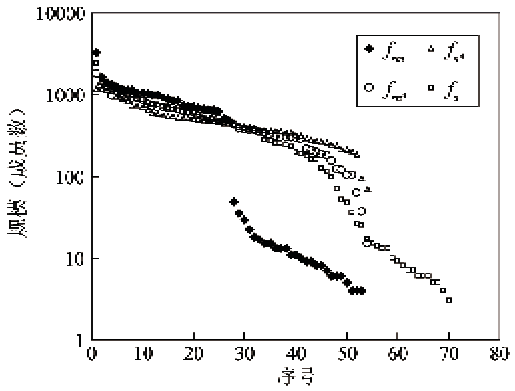

图 5 cm03 上聚类结果的规模分布

4 结 论

HAC 算法是迭代运行的，分析 HAC 算法和设计准则函数的关键是考虑每次合并对后续的合并产生的影响。一次合并会产生新的类，并使某些类的邻居减少。不断地合并邻居较少而共享邻居较多的类是提升 HAC 算法执行效率的关键。一次合并会改变后续合并时提高质量的程度，而优先合并较小的类有利于提高聚类结果的质量。使用 f_{dQ} 生成超大规模的类的主要原因是它以两个类之间的边数为主导，从而使一个较大的类不断合并其邻居。本文提出了准则函数 f_{n*} 和 f_{m*}，它们有效地控制合并的类的邻居数和共享邻居数，显著提高了 HAC 算法的

效率,并且在一定程度上提高了聚类结果的质量。由于计算类的共享邻居数耗费过多时间,f_{sn}。在提升 HAC 算法的执行效率方面比f_n。相差很多,而f_n。在提升结果质量上比f_m。相差很少,因此f_n。更适合处理较大规模的网络。根据实验还发现,聚类算法是决定最终聚类结果的分布的主要因素。

值得一提的是,一个准则函数在不同数据集上的加速比差别很大,这说明输入数据严重影响 HAC 算法的行为。另外,本文仅定性地考虑一个类的规模和类的邻居数、类内成员的度的和之间的关系,没有进行量化研究。上述问题都和复杂网络的数据模型有关,因此下一步研究的重点是根据输入数据的特征分析 HAC 算法。

参考文献

[1] Florek K, Lukaszewicz J, Perkal J, et al. Sur la liaison et la division des points d'un ensemble fini. *Colloq Math*, 1951, 2:282-285

[2] Zhao Y, Karypis G, Fayyad U. Hierarchical Clustering Algorithms for Document Datasets. *Data Min Knowl Discov*, 2005, 10(2):141-168

[3] Guha S, Rastogi R, Shim K. Rock: A robust clustering algorithm for categorical attributes. *Information Systems*, 2000, 25(5):345-366

[4] Karypis G, Han E-H, Kumar V. Chameleon: Hierarchical Clustering Using Dynamic Modeling. *Computer*, 1999, 32(8):68-75

[5] Clauset A, Newman M E J, Moore C. Finding community structure in very large networks. *Physical Review E*, 2004, 70(6):066111

[6] Wakita K, Tsurumi T. Finding community structure in mega-scale social networks. In: Proceedings of the 16th International Conference on World Wide Web, Banff, Alberta, Canada, 2007. 1275-1276

[7] Leon-Suematsu Y I, Yuta K. A framework for fast community extraction of large-scale networks. In: Proceeding of the 17th International Conference on World Wide Web, Beijing, China, 2008. 1215-1216

[8] Danon L, Diaz-Guilera A, Arenas A. Effect of size heterogeneity on community identification in complex network. *J Stat Mech*, 2006:P11010

[9] Shen H W, Cheng X Q, Guo J F. Quantifying and identifying the overlapping community structure in networks. *J Stat Mech*, 2009:P07042

[10] Newman M E J. Modularity and community structure in networks. *PNAS*, 2006, 103: 8577-8582

[11] Wang Y, An M. Effective Criterion Functions for Efficient Agglomerative Clustering on Very Large Networks. In: Proceedings of the 9th IEEE International Conference on Data Mining, Miami, USA, 2009. 1040-1045

[12] Kurita T. An Efficient Agglomerative Clustering-Algorithm Using a Heap. *Pattern Recognition*, 1991, 24(3): 205-209

[13] http://www.nd.edu/~networks/resources.htm, http://www-personal.umich.edu/~mejn/netdata/

[14] Brandes U, Delling D, Gaertler M, et al. On finding graph clusterings with maximum modularity. In: Proceedings of the 33rd International Conference on Graph-theoretic Concepts in Computer Science, Domburg, Germany, 2007. 121-132

[15] Arenas A, Danon L, Diaz-Guilera A, et al. Community analysis in social networks. *European Physical Journal B*, 2004, 38(2):373-380

[16] Yuta K, Ono N, Fujiwara Y. A Gap in the Community-Size Distribution of a Large-Scale Social Networking Site. *arXiv:physics/0701168v2*, 2007

Dynamic analysis of hierarchical agglomerative clustering algorithm and design of criterion functions

Wang Yang[*,**], Tu Dengbiao[***], An Mingyuan[*], Sun Ninghui[**], Wang Weiping[**]

([*]Graduate University of Chinese Academy of Sciences, Beijing 100190)

([**]Key Laboratory of Computer System and Architecture, ICT, CAS, Beijing 100190)

([***]National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing 100029)

Abstract

To improve the efficiency and the result quality of the hierarchical agglomerative clustering (HAC) algorithm, its dynamic analysis was conducted and the problem that how a merge influences the subsequent merges was studied, with the conclusions below: merging two clusters generates a new cluster, and reduces the number of neighbors of the shared neighbors of the two clusters; The new cluster and those clusters whose number of neighbors are decreased will be involved in the subsequent merges, and the efficiency will be influenced; A merge changes the value of the criterion function over the involved clusters and their candidate neighbors, and thus influences the quality of the subsequent merges. Based on the above analyses and the definition of modularity, existing criterion functions' influence on agglomeration and their limitations were investigated, and two new criterion functions were designed. The results of the experiments conducted based on many datasets show that the new criterion functions can improve the efficiency and the result quality of the HAC algorithm.

Key words: hierarchical agglomerative clustering (HAC) algorithm, criterion function, modularity, cluster analysis