一种基于加权多代表点的层次聚类算法*)

倪维健[1] 黄亚楼[1] 李 飞[2] 刘 赏[2]

(南开大学软件学院 天津300071)[1] (南开大学信息技术科学学院 天津300071)[2]

摘 要 CURE算法是一种凝聚的层次聚类算法,它首先提出了使用多代表点描述簇的思想。本文通过对已有的基于多代表点的层次聚类算法特点的分析,提出了一种新的基于多代表点的层次聚类算法WRPC。它使用了基于影响因子的簇代表点选取机制和基于k-近邻方法的小簇合并机制,可以发现形状、尺寸更为复杂的簇。实验结果表明,该算法在保证执行效率的情况下取得了更好的聚类效果。

关键词 层次聚类,代表点,k-近邻图,数据挖掘

An Agglomerative Hierarchical Clustering Algorithm Based on Weighted Representative Points

NI Wei-Jian[1] HUANG Ya-Lou[1] LI Fei[2] LIU Shang[2]

(College of Software, Nankai University, Tianjin300071)[1]

(College of Information Technical Science, Nankai University, Tianjin300071)[2]

Abstract As an agglomerative hierarchical clustering algorithm, CURE firstly employs the method of representing clusters by selecting some "representative points". Through the analysis of the feature of traditional hierarchical clustering algorithm, a novel agglomerative hierarchical clustering algorithm called WRPC is proposed in this paper. WRPC can identify clusters with complex shapes and avrious size by introducing the influence-weight-based representative points selection mechanism and k-nearest-neighbor-method-based clusters nesting mechanim. Experimental results show that WRPC can provide better clustering result with high executing efficiency.

Keywords Hierarchical clustering, Representative points, k-nearest neighbor graph, Data mining

1 引言

聚类分析是一种无监督的学习过程,其目标是基于一定的数据度量标准,把给定 d 维空间中的 n 个数据点聚集成 k 个簇,使簇内数据点的相似度极大化,簇间数据点的相似度极小化[1]。

聚类算法有多种,其中层次凝聚聚类算法是一个非常重要的分支,在实际应用中使用得比较广泛。传统的层次凝聚聚类算法有单链接和多链接方法[1],新近提出的层次凝聚聚类算法有CURE[2]、CHAMELEON[3]等。各种层次凝聚聚类算法的不同之处主要在于中间簇进行合并时使用的相似性度量不同。CURE提出了使用簇内的多个数据点来代表簇的思想,它首先选取每个簇内均匀分布的若干数据点作为这个簇的代表点,然后使用每个簇的代表点之间的最小距离来决定簇的合并,它的计算复杂度较小,对噪声具有较大的鲁棒性,并可以发现形状、尺寸较为复杂的簇;CHAMELEON在中间簇进行合并时考虑到了多种簇分布的情况,使用簇之间的相对互连性 RI 和相对近似性 RC 来描述簇间相似度,相比于CURE它可以得到质量更高的簇。

然而上述算法也存在着不足,CURE算法虽然机制较为简单,但无法在许多分布复杂的数据集中发现合理的簇;CHAMELEON算法虽然可以适应很多复杂的数据集,但其簇在合并时要在相关图算法[4]的基础上计算更多的相似度度量,实现较为复杂,时间代价较大,因此实用性相对较差。本文针对以上问题,提出了一种新的基于多代表点的层次聚类算法WRPC(Weighted Representative Points based Clustering),其核心思想是试图采用基于影响因子的代表点选取机制和基于 k-近邻的簇合并机制,从而在保证算法执行效率的前提下得到更好的聚类结果。

2 相关工作

2.1 基于多代表点的聚类算法CURE特点分析

CURE是一种基于多代表点的凝聚的层次聚类算法,它的整体框架是首先把每个数据点作为不同的簇,然后不断使用基于簇的代表点的方法对最相似的两个簇进行合并,直到簇的个数达到了用户指定的阈值。CURE的簇的合并过程分为两个阶段:第一阶段是代表点选取,即通过一定的策略选取簇内一定数量的点作为簇的代表点,第二阶段是小簇合并,即首先根据簇的代表点来计算簇之间的相似度,然后选取当前要合并的簇。这是整个CURE算法的核心和关键所在。

CURE中使用多代表点来描述簇的方法具有如下优点:第一,基于多代表点的簇间相似性度量既可以降低噪声点对簇合并的影响(噪声点不易被选为簇的代表点),又可以使相似性度量反映出簇的形状、分布等信息,因此得到的簇的质量更好;第二,在计算基于代表点的簇间相似度时,只须计算代表点之间的距离,而传统的簇间相似度是建立在簇内所有数据点之间的距离的基础上的,因此其计算的时间代价更小;第三,通过合理的选择机制得到的代表点可以最大程度地反映出簇的形状、位置、分布及密度等重要信息,从知识发现的角度讲,簇的多代表点是对簇所反映的知识的浓缩,即代表点既

*)本文得到教育部重点科学技术研究项目(02038)、天津自然科学基金项目(023600611)、南开大学亚洲研究中心资助。倪维健 硕士研究生,主要研究方向为数据挖掘和机器学习;黄亚楼 教授,博士生导师,主要研究方向为智能机器人系统、智能信息处理理论与技术。

可以描述簇本身的信息，又比原有的簇可以占有更少的存储空间。

然而 CURE 算法依然存在着一些不足，在很多情况下无法得到很好的聚类结果。

第一，在簇的形状、分布、密度不一致时，CURE 使用的代表点选取方法将可能选取不合理的代表点。在图1中描述的簇（数据集 $n=100$，代表点个数 $c=7$，“+”表示代表点），其左侧密度较大，右侧密度较小，但是如果使用 CURE 中的代表点选取方法，将在右侧密度较小的区域得到较多的点，显然在这种情况下，代表点的数量将无法反映簇内部的密度。究其原因，就是在选取代表点时，CURE 选取簇边缘的数据点作代表点的概率比较大，而在很多情况下簇边缘的点往往“代表性”并不大且成为噪声点的概率较大。

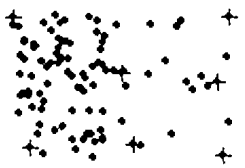


图1　簇密度不一致时 CURE 代表点选取情况

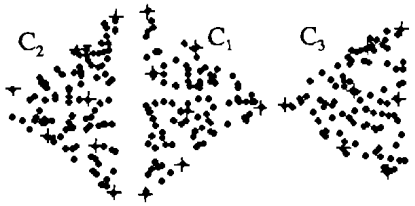


图2　簇形状特殊时根据代表点簇的合并情况

第二，CURE 在小簇合并时，计算两个小簇的相似度时仅考虑了这两个小簇的代表点之间的最小距离，这种策略使得簇的个别代表点可能较严重地影响簇间的相似度，而且这种相似性度量忽略了簇本身的部分信息，如图2（$n=300$，$c=7$，“+”表示代表点）所示的三个簇，如果使用这种合并策略，将把簇 C_1 和 C_3 进行合并，而实际上簇 C_1 和 C_2 的合并更加合理，因为虽然 C_1 和 C_3 的代表点之间的最小距离更小，但是 C_1 和 C_2 之间互相接近的数据点更多。这种问题在簇的形状更复杂、数据集维数较高的时候体现得更加明显。此外，如果在运行聚类算法之前对数据集进行了分块，或数据集具有不断增量的特性，要合并的小簇将产生互相重叠的现象，在这种情况下，簇的代表点之间的最小距离将无法度量簇之间的真实距离。

2.2　层次凝聚聚类算法 CHAMELEON 特点分析

作为一个重要的层次凝聚聚类算法，CHAMELEON 具有改进的簇的合并机制，它可以得到很好的聚类效果。它引入了两个指标来计算簇间相似度：相对互连性 RI 和相对近似性 RC[3]。其中，簇间相对互连性：

$$RI(C_i,C_j)=\frac{|EC_{(C_i,C_j)}|}{\frac{|EC_{C_i}|+|EC_{C_j}|}{2}}$$

即为簇间的绝对互连性（$|EC_{(C_i,C_j)}|$）与每个簇内部的连通程度（$|EC_{C_i}|$、$EC_{C_j}|$）平均值的比值，它主要描述了簇的形状信息在簇合并时的作用；簇间相对近似性：

$$RC(C_i,C_j)=\frac{\bar{S}_{EC(C_i,C_j)}}{\frac{|C_i|}{|C_i|+|C_j|}\bar{S}_{EC(C_i)}+\frac{|C_j|}{|C_i|+|C_j|}\bar{S}_{EC(C_j)}}$$

即为簇之间的绝对接近程度（$\bar{S}_{EC(C_i,C_j)}$）与每个簇内部数据点之间的接近程度（$\bar{S}_{EC(C_i)}$、$\bar{S}_{EC(C_j)}$）加权均值的比值，它主要描述了簇的密度信息在簇合并时的作用。这两个指标可以适应于多种数据集分布的情况，因此得到的簇的质量比 CURE 更好。但是 CHAMELEON 要先对数据集进行初始划分，然后对任两个小簇计算上述两个指标，每次计算都要遍历簇多遍，因此它实现比较复杂，且时间代价较大。

3　加权多代表点聚类算法 WRPC

针对上述问题，本文提出了一种新的加权多代表点的层次凝聚聚类算法 WRPC，它试图采用基于影响因子的簇代表点选取机制和基于 k-近邻方法的小簇合并机制，从而在保证执行效率的前提下取得更好的聚类效果。

3.1　基于影响因子的簇内多代表点的选取

选取簇的代表点是基于代表点的聚类算法的一个非常重要的环节，将直接影响到聚类得到的簇的质量。较合理的簇的代表点必须具有以下特征：

(i) 代表点在簇内的分布应是分散的，要避免多个代表点的聚集现象；

(ii) 代表点的分布要同数据集的分布一致，代表点的选取要由数据集的形状及密度等因素决定；

(iii) 噪声点应以小概率被选作代表点。

总之，一种好的代表点选取策略要使簇的代表点尽可能多地反映原簇的分布、形状、密度等信息。为此首先给出如下定义：

定义1（代表点的影响因子 IW）　设数据集中的簇 $C=\{d_1,d_2,\cdots,d_n\}$，其中 $d_i(1\leqslant i\leqslant n)$ 为簇内的数据点，其代表点为 $\{d_{p1},d_{p2},\cdots,d_{pc}\}$，根据每个数据点到代表点的最小距离可以得到 C 的一个划分 $\{C_1,C_2,\cdots,C_c\}$，其中的每个分块 C_i 与代表点 d_{pi} 一一对应，且分块 C_i 中的所有数据点距所有代表点中的 d_{pi} 最近。那么代表点 d_{pi} 的影响因子等于对应分块 C_i 的数据点个数在簇 C 中的百分比，即

$$IW(d_{pi})=\frac{|C_i|}{|C|}$$

图3描述了一个球形的簇（$n=300$，$c=16$）的代表点并对它们编号，表1给出了每个代表点的 IW。

在图3和表1中可以看出，对于数据集密集区域的代表点（1、10、11、14、15），它们对应的 IW 相对较大，而对于数据集稀疏区域的代表点（2、3、9、12）对应的 IW 则相对较小，由此，代表点的影响因子可以在一定程度上辨别数据集的噪声点，并反映数据集的密度分布。

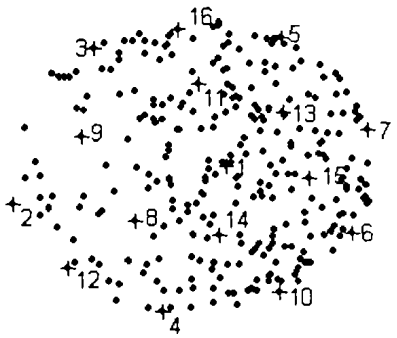


图3　球形簇得到的代表点及编号

表1　图1代表点的影响因子 IW

d_{pi}	1	2	3	4	5	6	7	8
$IW(10^{-2})$	9.0	2.7	4.7	4.7	6.3	5.7	5.0	6.0
d_{pi}	9	10	11	12	13	14	15	16
$IW(10^{-2})$	3.3	8.3	9.7	2.3	7.3	9.0	10.3	5.7

在得到簇的初始代表点后，WRPC 计算每个代表点的影

响因子，如果某代表点的影响因子小于用户指定的阈值 $\eta(0 \leqslant \eta \leqslant 1)$，那么将之从代表点集中删除，然后在簇中既没有被选作代表点又没有被从代表点集中删除的数据点中选取新的代表点来填补删除造成的空缺。这种调整过程不断重复，直到所有代表点的影响因子均大于 η。在我们的实验中，取 $\eta=\frac{1}{5*c}$ 时，调整过程得到的效果和效率均较好（其中 c 为簇代表点个数）。

在 WRPC 算法中，代表点的选取策略是：

(i)选取簇的中心点作为第一个代表点，因为簇的中心点的描述簇在数据集中的位置，与簇中其余的点相比可以表示更多的信息；

(ii)在簇中剩余的数据点中选取距已选代表点的最小距离最大的数据点作为一个新的代表点，这个过程不断迭代直到找到簇 c 个代表点；

(iii)如果存在影响因子小于指定阈值 η 的代表点，那么使用上述调整策略对初始代表点进行调整；

(iv)如果所有的代表点的影响因子都大于上述的最小阈值 η，那么迭代结束，这时已经找到所有合理的代表点。

上述步骤(ii)可以得到在簇内分散分布的初始代表点，通过步骤(iii)的调整，可以在代表点集合中排除噪声点及非常边缘的点，并且趋向于在簇密集区域选取更多的代表点。

3.2 基于 k-近邻图的小簇的合并

通过使用基于影响因子的小簇代表点选取方法，可以使得到的代表点可以很好地描述小簇的形状、位置、密度分布等信息，这为小簇的合并提供了大量的信息。WRPC 设计了一种基于 k-近邻的小簇合并方法。

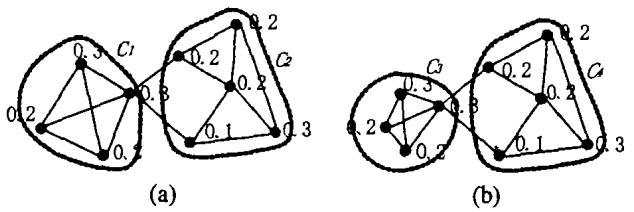


图4 4个簇上的2个3-近邻图

在进行小簇合并时，WRPC 首先使用了 k-近邻方法[5]对簇的代表点建模，然后根据如下的定义计算两个小簇间的相似度：

定义2(簇间相关度 $CR(C_i,C_j)$) 设两个簇 C_i 和 C_j，$RP(C_i)=\{d_{i1},d_{i2},\cdots,d_{ic}\}$ 是簇 C_i 的代表点集，$RP(C_j)=\{d_{j1},d_{j2},\cdots,d_{jc}\}$ 是簇 C_j 的代表点集，对簇 C_i 的代表点 d_{ik}，$IW(d_{ik})$ 表示 d_{ik} 的影响因子，$KNN(d_{ik})$ 表示簇 C_i 和簇 C_j 所有代表点集内代表点 d_{ik} 的 k 个最近的邻居的集合。如果点集

$V_i=\{v \mid v\in RP(C_i)\wedge((\exists\ u)u\in RP(C_j),u\in KNN(v))\}$

$V_j=\{v \mid v\in RP(C_j)\wedge((\exists\ u)u\in RP(C_i),u\in KNN(v))\}$

那么 $CR(C_i,C_j)=\sum_{v\in V_i\vee V_j} IW(v)$。

图4为在簇 C_1 和 C_2，C_3 和 C_4 上构建的3-近邻图，根据定义2，C_1 和 C_2 的相关度 $CR(C_1,C_2)=0.2+0.1+0.3=0.6$，$C_3$ 和 C_4 的相关度 $CR(C_3,C_4)=0.2+0.1=0.3$（在(a)图中，簇 C_2 中存在簇 C_1 中影响因子为0.3的代表点的3-近邻，而在(b)图中，簇 C_4 中不存在簇 C_3 中影响因子为0.3的代表点的3-近邻）。

在得到簇间的相关度后，在每次小簇的合并时都选出相关度最大的两个簇进行合并，然后计算新簇的最大相关度，并调整与这两个簇最相关的簇的新的最大相关度。

使用簇间相关度来表示簇间的相似度具有如下特点：首先，簇间相关度使用 k-近邻方法可以动态捕捉代表点的邻域，使簇间相似度不仅与簇之间的连接情况有关，而且还与簇内的互连情况有关，用较简单的方法把 CHAMELEON 中“相对”的概念引入到簇间相似度中，这将使簇的合并更加合理，比如图4所示的情况，虽然 C_1 和 C_2、C_3 和 C_4 簇间的绝对距离接近，但是簇间的相关度却是两个密度接近的簇 C_1 和 C_2 较大；其次，簇间相关度的计算使用了相关代表点的影响因子，而影响因子反映了其所在位置簇内的密度，所以它在簇合并时考虑到了簇的密度信息；再次，WRPC 只在簇的代表点上构建 k-近邻图，这使得所得到的 k-近邻图的规模较小，从而使算法具有较低的时间代价。

3.3 算法框架及伪码实现

WRPC 在开始时把每个数据点作为单独的簇，然后不断对最相近的两个簇进行合并，并对合并后的簇重新选取新的代表点（如果此时簇中包含的数据点个数小于代表点个数 c，那么直接把簇中的所有点作为这个簇的代表点）。这个过程不断迭代直到任意簇之间的相似度达到了用户指定的上限或全部数据点被合并成一个簇。在合并过程中，如果小簇的增长缓慢，那么将其作为噪声点排除。WRPC 算法的整体流程见表2。

表2 WRPC 整体流程的伪代码

```
WRPClustering(D,c,dmin)
//输入数据集 D,簇代表点数 c,簇间相关度下限 dmin,
//返回簇列表 CL
{ //构造初始簇列表 CL
  CL=InitalizeClusters(D);
  max=MAX_CR;
  while (max>dmin)&(CL.count>1)
  { //选择最相关的簇 Ci 和 Cj 合并成簇 Cnew
    [Cnew,max]=SelectandMerge(Ci,Cj);
    if (|Cnew|>c)
      GetRP(Cnew);//获取新簇 Cnew 的代表点
      //更新簇列表及每个簇的 nearestCR 和 nearestC
      UpdateCL(CL,Cnew,Ci,Cj);
  }
  RecordLabel(D);//标记所有记录的簇归属
  return CL;
}
```

其中，代表点选取阶段的流程见表3，使用的数据结构如下：

```
struct Point //数据点
{ int id;//数据集中的记录号
  int type;//类型(0:可选做代表点的数据点,1:已被选作
           //代表点的数据点,2:不可选做代表点的数据点)
  }
struct RPoint //代表点
{ int id;//数据集中的记录号
  float iw;//影响因子
}
```

在簇的合并阶段，对于每个簇均保存与之最相关的簇及其对应的相关度。在进行了簇的合并后，如果由簇 C_i 和 C_j 合并得到簇 C_{new}，那么要首先计算出 C_{new} 最相关的簇及对应的相关度，然后对那些以前与 C_i 或 C_j 最相关的簇的相关参数进行更新。算法流程见表4，使用的数据结构如下：

```
strcut Cluster //簇
{ int id;//簇标识符
  Cluster * nearestCluster; //与之最相关的簇
  float nearestCR;//与最相关簇间的相似度
  RPointList RPList;//指向簇内的每个代表点
  PointList records;//指向簇内的每个数据点
}
```

表3 WRPC 代表点选取过程的伪代码

```
GetRP(C,η)
//输入簇 C 和代表点调整阈值 η
{ C.RPList=null;//初始化代表点列表
  C.RPList.add(C.mean)
  rpCount=c-1;//还要选取的代表点的数量
  needAdjust=true; adjustCount=0;
  while (needAdjust)
  { for(i=0;i<rpCount;i++)
    { //计算距已选代表点最小距离最大的
      //点作为下一个代表点并添加至代表点列表
      rp=NextRP(C);
      C.RPList.add(rp);
    }
    AssignPoint(C);//根据当前代表点分配数据点
    needAdjust=false; rpCount=0;
    foreach RPoint rp in C.RPList
    { if (rp.iw<η)//需要调整
      { C.RPList.delete(rp);
        needAdjust=true; rpCount++;
        Label(rp);//标记 rp 不可再被选作代表点
      }
    }
    if ((needAdjust)&(++adjustCount>MAX_ADJ))
        needAdjust=false;//防止 η 不当造成死循环
  }
  return RPList;
}
```

表4 WRPC 的簇列表更新过程的伪代码

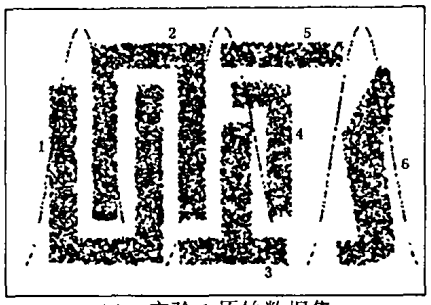
```
UpdateCL(CL,Cnew,Ci,Cj)
//输入簇列表 CL、Cnew,返回更新后的簇列表
{ Cnew.nearestCR=0;
  Cnew.nearestCluster=null;
  foreach Cluster C in CL
  { //仅对与簇 Ci 或 Cj 相关的簇调整其
    //nearestCluster 和 nearestCR
    if ((C.nearestCluster=Ci∨Cj)|
        (C=Ci.nearestCluster)|(C=Cj.nearestCluseter))
    { //在 C.RPList 和 Cnew.RPList 上构建 k-近邻图
      G=KNNGraphCreate(C,Cnew);
      curCR=GetCR(G);//计算 G 的相关度
      if(curCR>C.nearestCR)
      { C.nearestCluster=Cnew;
        C.nearestCR=curCR;
      }
      if(curCR>Cnew.nearestCR)
      { Cnew.nearestCluster=C;
        Cnew.nearestCR=curCR;
      }
    }
  }
  CL.add(Cnew); //在 CL 中添加新簇 Cnew
  return CL;
}
```

4 实验

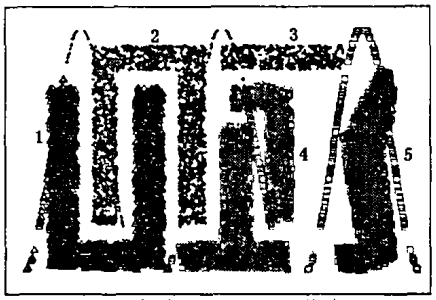
为了验证 WRPC 算法的性能,我们进行了若干实验。由于 WRPC 主要针对 CURE 等算法进行了改进,因此我们在实验中侧重于 WRPC 和 CURE 及 CHAMELEON 算法的比较。实验的硬件环境是 Celeron 1.7G CPU,256MB DDRRAM 的 PC,软件环境为操作系统 Windows 2000 Server,编程语言 c#。另外,WRPC 使用的噪声点去除策略类似于 CURE 中的方法,其效果相当,所以实验中没有考虑噪声点去除问题。

4.1 聚类效果分析

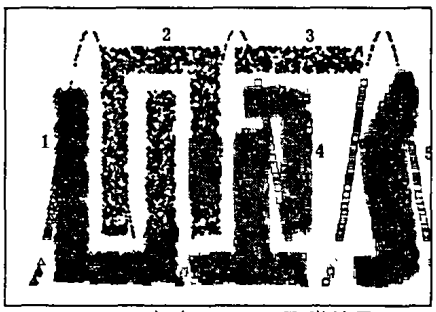
实验1:这个实验的目的是比较在簇的分布比较复杂时 WRPC 和 CURE 算法所得到的簇的质量,实验数据集为模仿 CHAMELEON 算法[3]中的实验数据集 DS3生成的二维数据集,如图5(a)所示,其中数据点个数为8000,包含6个原始簇,其形状、大小、位置变化均比较大,另外存在类似于余弦分布的噪声点,为了验证算法的效果,比起 CHAMELEON 中的 DS3,我们增加了这部分噪声数据的数量,加强了噪声的干扰性。参数设置为:CURE 中的收缩因子 $\alpha=0.35$,代表点个数 $c=20$,WRPC 算法中的 IW 调整下限为2.5%,k-近邻图中 $k=8$,代表点个数 $c=20$。图5(b)和(c)分别为 WRPC 和 CURE 在凝聚到5个簇时得到的聚类结果,我们对这些簇进行了标号,并用三种颜色和三种图形的组合表示得到的簇。



(a) 实验1原始数据集



(b) 实验1 WRPC 聚类结果



(c) 实验1 CURE 聚类结果

图5 实验1的数据集及实验结果

由图5可见,WRPC 得到的5个簇中(图5(b)),簇1、2、3、5与理想中的簇基本吻合,由于数据集的余弦噪声干扰过强,簇4合并了理想中的簇3、4(图5(a));CURE 得到的5个簇中(图5(c)),簇3、5与理想中的簇基本吻合,而簇1、2、4与理想中的簇有较大的误差。CHAMELEON 可以得到接近于图5(a)中的簇。

因此,当簇的位置、大小、形状变化比较复杂时,WRPC 比 CURE 可以得到更好的聚类效果。

4.2 WRPC 效率分析

实验2:这个实验主要是为了比较 WRPC 和 CURE、CHAMELEON 的执行效率,实验数据集为模仿 CURE 算法[2]的实验中的 DS1(图6),这三个算法可以在这个数据集上得到相似的正确结果,数据点个数由10000递增至50000。为了实现的简单化,我们使用线性数据结构实现了 CURE 和 WRPC 算法,如果它们在距离计算和排序时均使用树型结构,执行时间将更少。WRPC 与 CURE 参数设置同实验1,CHAMELEON 算法[3]中 k-近邻图中 $k=10$,第一阶段数据集划分中 MINSIZE=2%,簇间相似度度量中的 RC 的指数 $\alpha=2$。图7表示了 WRPC 与 CURE、CHAMELEON 在执行时间上的比较。

实验2可以说明,在执行时间上,WRPC 与 CURE 比较接近,这两个算法的时间复杂度是一致的。但是,WRPC 的执行时间要远小于 CHAMELEON,因此,WRPC 具有更高的执行效率。

经过上述实验,可以看出 WRPC 算法在和 CURE 算法

具有接近的时间代价的情况下，取得了更好的聚类效果，因此，WRPC 算法在聚类效果和时间代价上取得了较好的折衷，实用性更好。

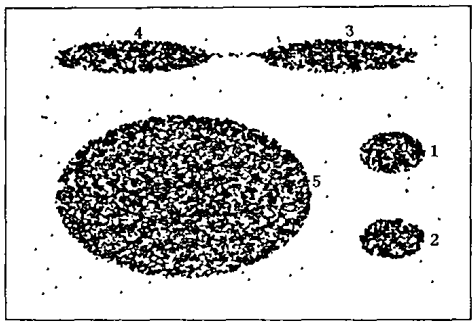


图6 实验2数据集

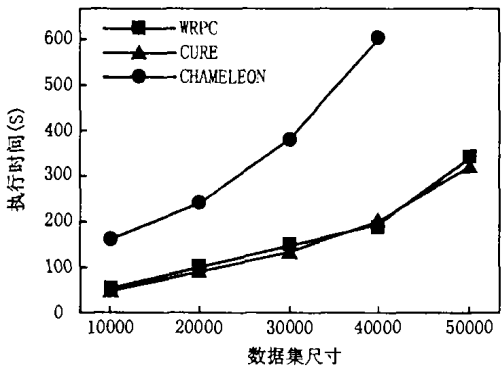


图7 实验2算法执行时间对比

结束语 本文对基于多代表点的聚类方法进行了研究，较为深入地讨论部分已有方法的优点和存在的不足，并在此基础上提出了 WRPC 算法，该算法采用了基于影响因子的代表点选取机制和基于 k-近邻方法的小簇合并机制。实验结果表明，WRPC 在时间代价与 CURE 相同的前提下，得到了接近于 CHAMELEON 的聚类效果，因此在算法的时间代价和聚类效果方面取得了更好的折衷。由于多代表点可以作为簇所反映的知识的浓缩，因此我们将对基于多代表点的可继承聚类算法做进一步的研究。

参 考 文 献

1 Kantardzic M. Data Mining concepts, models, method, and algorithms. 闪四清等译. 数据挖掘 概念、模型、方法和算法. 清华大学出版社，2003
2 Guha S, Rastogi R, Shim K. CURE: an efficient clustering algorithm for large databases. In: Proc. of 1998 ACM-SIGMOD Int. Conf. Management of Data, 1998
3 Karypis G, Han E-H, Kumar V. CHAMELEON: a hierarchical clustering algorithm using dynamic modeling. COMPUTER, 1999, 32: 68～75
4 Karypis G, Kumar V. Mulitilevel k-way hypergraph partitioning. In: Proc. of the Design and Automation Conf., 1999
5 Jarvis R A, Patrick E A. Clustering using a similarity measure based on shared nearest neighbors. IEEE Trans on Computers, 1973, C-22(11)
6 O'Callaghan L, Mishra N, Meyerson A, Guha S, Motwani R. Streaming-Data algorithms for high-quality clustering. In: 18th Intl. Conf. on Data Engineering, 2002
7 Ng R T, Han Jiawei. Efficient and effective clustering methods for spatial data mining. In: Proc. of the 20thVLDB Conf. Santiago, Chile, 1994
8 Li C R, Chen M S. A robust and efficient clustering algorithm based on cohesion self-merging. SIGKDD, Edmonton, Alberta, Canada, 2002

(上接第146页)

```
holds(P&Q,S):- holds(P,S),holds(Q,S).
holds(PVQ,S):- holds(P,S);holds(Q,S).
holds(P⇒Q,S):- holds(-PVQ,S).
holds(P⇔Q,S):- holds((P⇒Q)&(Q⇒P),S).
holds(-(-P),S):- holds(P,S).
holds(-(P&Q),S):- holds(-PV-Q,S).
holds(-(PVQ),S):- holds(-P&-Q,S).
holds(-(P⇒Q),S):- holds(-(-PVQ),S).
holds(-(P⇔Q),S):- holds(-((P⇒Q)&(Q⇒P)),S).
holds(-all(X,P),S):- holds(some(X,-P),S).
holds(-some(X,P),S):- not holds(some(X,P),S).
holds(-P,S):- isAtom(P),not holds(P,S).
holds(all(X,P),S):- holds(-some(X,-P),S).
holds(some(X,P),S):- sub(X,_,P,P1),holds(P1,S).

/* Restore Situation arguments to situation suppressed terms */
holds(A,S):-reSitArg(A,S,F),F;not reSitArg(A,S,F), isAtom(A).
isAtom(A):- not( A=-W;A=(W1&W2); A=(W1VW2); A=(W1=>W2); A=(W1<=>W2); A=some(X,W); A=all(X,W)).
reSitArg(capa(A),S,capa(A,S)).
reSitArg(start(T),S,start(S,T)).
reSitArg(now(T),S,now(S,T)).

/*"now" is a synonym for "start"*/
now(S,T):-start(S,T).
/*the start time of situation trans(A,B) is the time of A*/
start(trans(A,S),T):-timer(A,T).
```

总结 面向 agent 的理论与技术是当今人工智能与软件工程研究的重要内容，agent 理论、agent 结构、面向 agent 程序设计语言(AOPL)是 agent 理论与技术研究的核心内容[7]。

AOPLID 是一种建立在情境演算理论基础上的面向 agent 程序设计语言。与其它相关工作相比，该语言既有较完善的理论基础又有完整的程序设计语言的面向 agent 程序设计的解决方案[2]。本文主要对 AOPLID 进行时序扩充，使得 AOPLID 语言能够表示并处理带时间参数的并发行动。首先，在经典的情境演算中引入时间变元，将持续动作看成具有瞬时开始和瞬时终止的过程，从而在时序情境演算中表达带时间参数的并发行动；其次，对离线方式下 AOPLID 程序语义进行了适当改造，设计并实现了时序 AOPLID 离线解释器。

与此同时，在 agent 理论与 agent 结构研究方面，我们基于情境演算理论，提出了基于情境演算的智能体结构和刻画 agent 自主性的框架[8,9]；基于关系理论，提出了基于关系的两维意向结构[10]。

今后的工作主要有进一步扩充 TAOPLID 语言，研究如何在 TAOPLID 中表示其它 agent 的知识、不确定性的知识并进行相应推理；研究在 TAOPLID 中多 agent 规划的各种表示方法及其解释器的实现。

参 考 文 献

1 张东摩. 面向 Agent 的智能系统的理论研究：[博士后出站研究总结报告]. 南京大学研究生院，1998
2 郭磊，戈也挺，陈世福，张东摩. 一种意向驱动式面向 agent 程序设计语言. 软件学报，2003，14(3)：383～391
3 McCarthy J, Hayes P. Some Philosophical problems from the Standpoint of Artificial Intelligence. In: Meltzer B, Michie D. eds. Machine Intelligence 4, Edinburgh University Press, Edinburgh, Scotland, 1969. 463～502
4 Reiter R. Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems. Cambridge, MA: MIT Press, 2001. 35～105
5 Levesque HJ, Reiter R, Lesperance Y, et al. GOLOG: A logic programming language for dynamic domains. Journal of Logic Programming, 1997, 31: 59～84
6 Reiter R. Sequential Temporal GOLOG. In: Proc. of KR'98, 1998. 547～556
7 刘大有，杨鲲，陈建中. Agent 研究现状与发展趋势. 软件学报，2000，11(3)：315～321
8 李斌，吕建，朱梧槚. 基于情境演算的智能体结构. 软件学报，2003，14(4)：733～742
9 李斌，陈韬略，吕建. 一个刻画 Agent 自主性的框架. 南京大学学报，2004，40(2)：137～145
10 李斌，朱朝晖，陈韬略，吕建，朱梧槚. 基于关系的两维意向结构. 软件学报，2004，15(4)：512～521