可间断运行的 K-means聚类算法*

黄志华[1,2a], 温步瀛[2b], 王国乾[3]

(1 厦门大学 信息科学与技术学院, 福建 厦门 361005; 2 福州大学 a 数学与计算机科学学院; b 电气工程与自动化学院, 福州 350108; 3 福建省计算中心, 福州 350003)

摘 要: 引入事务的恢复机制改进 K-means算法,改进后的算法允许在运行过程中的任何时刻停机,重新启动后可在停机前运算成果的基础上继续运算,直至算法结束。改进后的算法使得普通机器条件下针对大数据集运用 K-means算法成为可能。改进后的算法在长达 400 h的聚类运算中得到了检验。

关键词: K-means算法; 聚类; 恢复机制

中图分类号: TP18 文献标志码: A 文章编号: 1001-3695(2009)06-2053-03

doi 10 3969/j issn 1001-3695 2009 06 016

Recoverable implementation of K-means clustering algorithm

HUANG Zhi-hua[1,2a], WEN Bu-ying[2b], WANG Guo-qian[3]

(1 *School of Information Science & Technology, Xiamen University, Xiamen Fujian 361005, China; 2 a. College of Mathematics & Computer Science, b. College of Electrical Engineering & Automation, Fuzhou University, Fuzhou 350108, China; 3 Computing Center of Fujian Province, Fuzhou 350003, China*)

Abstract This paper improved K-means clustering algorithm by using transaction recovery mechanism. The new algorithm was able to resume itself without loss of computing time after the computer running it was shut down on purpose or by chance at any time, so that it could achieve its goal in big data sets on ordinary computers. It was verified in a clustering task which spent as long as 400 hours.

Key words K-means algorithm; clustering; recovery mechanism

1 聚类算法的研究与应用

聚类分析有着几十年的研究历史,众多聚类算法相继被提出。聚类已成为模式识别、数据挖掘等研究方向的重要研究内容,在识别数据的内在结构方面具有极其重要的作用。聚类分析在众多应用领域发挥了重要作用。例如,在图像处理中,聚类可被用于图像分割,同时也是图像特征提取的重要手段;聚类经常在各种应用领域中被用做数据压缩或数据降维的手段。聚类还应用于统计科学,对生物学、心理学、考古学、地质学、地理学、市场营销等研究都有重要作用[1]。

文献[1]把聚类算法分为层次聚类算法、划分式聚类算法、基于密度和网格的聚类算法,以及其他聚类算法四种类型。层次聚类算法又称为树聚类算法,它使用数据的联结规则,透过一种层次构架方式,反复将数据进行分裂和聚合,以形成一个层次序列的聚类问题解。划分式聚类算法需要预先指定聚类数目或聚类中心,通过反复迭代运算,逐步降低目标函数的误差值,当目标函数值收敛时,得到最终聚类结果,K-means聚类算法就是该类型中最具代表性的算法。基于网格和密度的聚类算法在以空间信息处理为代表的众多领域中有着广泛应用,它们通过数据密度,或使用网格,或两者结合使用来发现类簇。其他聚类算法指近年来研究者提出的新颖的聚类算法,如采用具有不同偏好的蚁群系统来解决数据分类的问题。

聚类算法在得到广泛应用的同时也面临着一些问题。大多数聚类算法都需要预先给定参数,如果没有相关知识和经验,合理给出聚类算法的参数是不可能的。对于层次聚类算法,如何找到聚合或分裂过程的有效终止条件仍然是一个开问题。对于划分式聚类算法,如何快速找到类的合理个数和较好的初始类中心点集仍然是很有价值的研究课题。聚类算法的聚类结果有一定的不可预见性,同一算法对不同的数据集聚类效果有差异,不同算法对同一数据集聚类效果也存在差异,即使同一聚类算法在同一数据集上也会因为相似性度量方式不同而产生不同的聚类结果。

2 K-means聚类算法的问题

K-means聚类算法是一种划分式聚类算法,由 MacQueen于 1967年首次提出。该算法是一个非常经典且被广泛应用的聚类方法。其思想如下:在 d维欧式空间中给定 n个数据点组成的集合 P,指定一个整数 k,要求在 d维空间中找到一个由 k个点组成的集合,称为 centers,使得 P中的每个点到离它最近的 center之间的欧式距离的均值最小[2]。解决 K-means问题是非常困难的,针对该问题,目前未见有效的解决方案被提出。Lloyd的算法就是一个解决 K-means问题的近似方法,通常所说的 K-means聚类算法指的就是 Lloyd的算法。该算法可描述如下:

收稿日期: 2008-09-22; 修回日期: 2008-11-12 基金项目: 福建省自然科学基金资助项目(2008J0018)

作者简介: 黄志华(1971-),男,湖北黄石人,副教授,博士研究生,主要研究方向为智能信息处理、数据库技术、操作系统等(hzh@fzu.edu.cn);温步瀛(1967-),男,副教授,博士,主要研究方向为电力系统优化运行、电力市场、风电并网运行技术;王国乾(1971-),男,福建长汀人,工程师,博士研究生,主要研究方向为分布式系统、网格计算、数据库技术、智能信息处理等.

a)随机指定 k个聚类中心 $C(c_1, c_2, \cdots, c_k)$;

b)对 P中的每个点 x_i, 找到离它最近的聚类中心 c_r, 并将其分配到 c_r 标注的类中, 同时计算 $D = 1/n\sum_{i=1}^{n}(\min_{r=1,\cdots,k} d(x_i, c_r))$, 如果 D 值收敛, 转步骤 e);

c)将每个 $c_r(r= 1, 2, \cdots, k)$更新为它所标注的类的中心;

d)转步骤 b);

e)返回 $c_1, c_2, \cdots, c_k$ 并终止本算法。

该算法描述简单, 易于实现, 是最为广泛应用的聚类算法之一。但是, 该算法仍然存在三个较为突出的问题有待进一步研究改进。第一, 对于一个给定的集合 P, 整数 k 指定为多少才是合理的; 第二, 该算法可能终止于 D 的一个局部极小值, 而不是 D 的最小值; 第三, 该算法的计算代价非常大, 需要耗费很长的计算时间, 往往不可能在大数据集上运用该算法。

针对第一个问题, 文献 [3]进行了有益的研究, 提出一种基于层次思想的计算方法。它首先扫描数据集获得统计值, 然后自底向上地生成不同层次的数据划分, 增量地构建一条关于不同层次划分的聚类质量曲线, 曲线极值点所对应的划分用于估计最佳划分数。

针对第二个问题, 文献 [2, 4]分别从不同的角度进行了研究。文献 [2]以交换 centers为基础提出了不同于 Lloyd算法的求解 K-means问题的近似算法; 文献 [4]通过改变 K-means聚类算法的步骤 a), 在该步骤中计算得到一个优质的聚类中心点集, 以此点集作为初始中心点集启动 K-means聚类算法, 从而使得 K-means聚类算法能够收敛到一个较好的局部极值。

针对第三个问题, 文献 [5]提出了一种能够降低计算代价的方法。K-means聚类算法的计算代价主要集中在步骤 b)。文献 [5]提出了一种称为 k-d树的数据结构来组织集合 P, 使得 K-means聚类算法每次执行步骤 b)时, 不用计算 P中每个点和 centers中每个点的距离, 减少了计算距离的计算量。文献 [6]对文献 [5]的方法进行了细小的改进, 并通过深入分析和大量的实验说明了该方法确实在一定的条件下能够提高 K-mean聚类算法的运行效率。

K-d树是一个二叉树。该树的根节点关联着一个超立方体, 该超立方体是能够包含 P中所有点的最小超立方体。K-d树的每个非根节点也关联着一个超立方体, 该超立方体是其父节点所关联的超立方体被一个超平面分割得到的, 其父节点所关联的超立方体被一个超平面分割成两部分, 一部分被左孩子节点继承, 另一部分被右孩子节点继承。分割超平面可以由不同的方法来产生, 一个简单可行的方法是选择与超立方体中所有点跨度最大的坐标轴正交且通过所有点在该坐标轴的中值的超平面。K-d树的叶节点关联的超立方体所包含的点数小于等于一个规定值, 该规定值最小为 1。K-d树在每次聚类前被构建, 聚类过程中没有变化。这里将文献 [5, 6]中的算法称为高效 K-means聚类算法。

高效 K-means聚类算法如下:

a)随机指定 k个聚类中心 $C(c_1, c_2, \cdots, c_k)$, 针对集合 P构建 k-d树, 根节点为 R;

b)调用 filter(R, C), 同时计算 $D = 1/n\sum_{i=1}^{n}(\min_{r=1,\cdots,k} d(x_i, c_r))$, 如果 D 值收敛, 转步骤 e);

c)将每个 $c_r(r= 1, 2, \cdots, k)$更新为它所标注的类的中心;

d)转步骤 b);

e)返回 $c_1, c_2, \cdots, c_k$ 并终止本算法。

其中: filter(R, C)是一个递归函数。先根遍历 R所标示的 k-d树, 遍历过程中把根节点与 C关联, 其他节点首先从父节点继承聚类中心集; 然后根据该节点所关联超立方体的几何特性过滤掉部分聚类中心, 剩下的聚类中心构成的集合就是此节点关联的聚类中心集合。遍历过程遇到叶节点或节点关联的聚类中心集合仅包含一个聚类中心时, 计算该节点所关联超立方体中每个点与该节点所关联聚类中心集合中每个聚类中心的距离, 对于每个点都找到离它最近的聚类中心 c_r, 并将其分配到 c_r 标注的类中。Filter(R, C)的详细表述参见文献 [6]。在为每个点寻找最近的聚类中心时, filter(R, C)避免了计算每个点与所有聚类中心间的距离, 而只是计算每个点与可能的聚类中心间的距离, 从而减少了计算距离的量。其代价是在聚类前需要构建 k-d树, 在每次迭代时需要遍历 k-d树并在遍历的过程中过滤无关的聚类中心。总体而言, 高效 K-means聚类算法降低了 K-means聚类的计算代价[6]。

针对第三个问题, 本文从另一个角度寻求解决方法。由于有时待聚类的数据集非常大, 即使高效 K-means聚类算法也要运行很长时间。普通的 PC机通常难以承受过长时间的连续运行负载。当在普通的 PC机上针对大数据集进行 K-means聚类时经常会遇到死机、意外关机或因故需要关机等情况。当需要的运算时间很长时, 几乎每次启动 K-means聚类算法都会遇到这样的情况。在这样的背景下, K-means聚类算法在事实上是不可行的。本文引入事务的恢复机制改进 K-means聚类算法, 使得它在运行过程的任何时刻都可以停机, 系统重启后, 可在停机前运算成果的基础上继续运行。改进后的算法不要求一次性完成 K-means聚类计算, 允许很长的聚类运算间断地在普通的 PC机上完成。

3 改进的 K-means聚类算法

3.1 事务的恢复机制

事务是交给计算机的一组不可分割的操作序列, 该组操作序列要么全部完成, 要么全部没有做。事务的概念出现在计算机的各个领域。例如, 在数据库领域, 事务是数据库系统处理数据的基本单位, 操作系统通常用事务的方式实现删除文件、移动文件等操作, 较好的软件安装工具把软件安装过程组织成事务。事务的不可分割性也常被称为原子性, 是靠恢复机制来实现的。事务的恢复机制可简单地概括为回滚和重做, 如果系统在某一时刻崩溃, 有的已完成事务的处理结果可能丢失, 正在运行的事务只有部分操作被完成, 系统重启后, 要重做那些丢失了结果的已完成事务, 回滚当时正在运行的事务。重做和回滚都是依据日志来完成的, 日志记录着事务的每个操作信息, 日志中的信息在每个事务的每个操作执行前被记入。

3.2 K-means聚类算法的恢复机制

为了让 K-means聚类算法可以间断地运行, 需要为 K-mean聚类算法建立一套恢复机制。无论 K-means还是高效 K-means聚类算法, 都是迭代算法, 每次迭代过程仅依赖于上一次迭代的结果。它们在运行过程的任何时刻停机, 只要把停机时刻的上一次迭代结果作为聚类中心 C的初值重新运行即可。参照事务的恢复机制, 为 K-means聚类算法增加一个运行日志文件, 专门记录以前迭代的结果。当 K-means聚类算法在运行过程中停机后, 可从日志文件中读出上一次迭代的结果,

以此作为聚类中心 C的初值重新启动 K-means聚类算法。这样, K-means聚类算法实际上就是在停机前运算成果的基础上继续运行。如果把每次迭代过程看成一个事务, 上述效果相当于重做了已完成事务回滚的未完成事务。

为实现以上机制, K-means聚类算法的运行日志文件应能够可靠地提供当前运行时刻的上一次迭代结果。在日志文件中保存以前每次迭代的结果可以达到此目标, 但是, 很显然, 保存以前每次迭代的结果没有必要。在日志文件中仅保存上一次迭代结果不能确保可靠地提供当前运行时刻的上一次迭代结果。因为在这样的情况下, 每次向日志文件写入上一次迭代结果必然覆盖再上一次的迭代结果, 该覆盖也是一个需要经历一定时间的过程, 如果停机恰恰发生在这个过程中, 日志中的信息将是混乱的, 既不是上次迭代的结果也不是再上一次的迭代结果, 这样的信息对于恢复 K-means聚类算法的运行是没有意义的。

本文采用的方法是让日志文件保存最近两次的迭代结果, 当向日志文件写入新的迭代结果时覆盖日志文件中较早的那次迭代结果。如果停机发生在覆盖的过程中, 当前迭代并未完成, 日志中出现混乱的部分是较早的那次迭代结果, 恢复运算所需要的较新的那一次迭代结果是完好的。日志文件的结构如图 1所示。其中: finish是一个长整数, 记录当前已完成的迭代次数, 其初值为 0; C_1和 C_2用于保存最近两次的迭代结果, 其结构与聚类中心 $C(c_1, c_2, \cdots, c_k)$一致, 对于一个具体的聚类任务, C_1和 C_2的结构及其大小都是固定不变的。

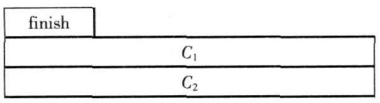

图 1　日志文件结构

写日志的过程可表述如下:

a)读 finish到 cur;

b) i= cur% 2+ 1;

c)将当前迭代得到的聚类中心集合 C写入 C_i;

d) cur= cur+ 1;

e)将 cur写入 finish, 结束。

当需要依据日志恢复时, 首先读取 finish, 如果 finish等于 0, 表明停机前并未成功完成一次迭代, 不必恢复, 直接重新启动算法; 否则, 判断 finish为奇数还是偶数。若为奇数, 读取 C_1; 若为偶数, 读取 C_2, 将读取的内容作为聚类中心集合 C的初值来启动算法, 从而达到恢复运行的目的。

3.3　改进后的算法

本文将引入恢复机制的 K-means聚类算法称为可间断运行的 K-means聚类算法, 它可被表述如下:

a)若为非恢复模式, 随机指定 k个聚类中心 $C(c_1, c_2, \cdots, c_k)$, 并创建日志文件, 赋值给 finish为 0, 转步骤 d), 否则转步骤 b);

b)若为恢复模式且 finish为 0, 随机指定 k个聚类中心 $C(c_1, c_2, \cdots, c_k)$, 转步骤 d), 否则转步骤 c);

c)判断 finish为奇数还是偶数, 若为奇数, 读取 C_1, 若为偶数, 读取 C_2, 将读取的内容赋值给聚类中心 $C(c_1, c_2, \cdots, c_k)$;

d)对 P中的每个点 x_i, 找到离它最近的聚类中心 c_r, 并将其分配到 c_r标注的类中, 同时计算 $D = 1/n\sum_{i=1}^{n}(\min_{r=1,\cdots,k} d(x_i, c_r))$, 如果 D值收敛, 转步骤 g), 否则转步骤 e);

e)将每个 $c_r(r= 1, 2, \cdots, k)$更新为它所标注的类的中心, 调用写日志过程;

f)转步骤 d);

g)返回 $c_1, c_2, \cdots, c_k$并终止本算法。

可间断运行的 K-means聚类算法有两种运行模式, 即非恢复模式和恢复模式。非恢复模式指该算法第一次运行一个聚类任务; 恢复模式指该算法已运行过此聚类任务, 此次运行是在上次运行的基础上恢复运行, 此次运行的模式作为该算法的参数由调用者指定。

高效 K-means聚类算法也被改进为可恢复的形式, 本文称之为可间断运行的高效 K-means聚类算法。它可被表述如下:

a)若为非恢复模式, 随机指定 k个聚类中心 $C(c_1, c_2, \cdots, c_k)$, 针对集合 P构建 k-d树, 根节点为 R, 并创建日志文件, 赋值给 finish为 0, 转步骤 d), 否则转步骤 b);

b)若为恢复模式且 finish为 0, 随机指定 k个聚类中心 $C(c_1, c_2, \cdots, c_k)$, 转步骤 d), 否则转步骤 c);

c)判断 finish为奇数还是偶数, 若为奇数, 读取 C_1, 若为偶数, 读取 C_2, 将读取的内容赋值给聚类中心 $C(c_1, c_2, \cdots, c_k)$;

d)调用 filter(R, C), 同时计算 $D = 1/n\sum_{i=1}^{n}(\min_{r=1,\cdots,k} d(x_i, c_r))$, 如果 D值收敛, 转步骤 g), 否则转步骤 e);

e)将每个 $c_r(r= 1, 2, \cdots, k)$更新为它所标注的类的中心, 调用写日志过程;

f)转步骤 d);

g)返回 $c_1, c_2, \cdots, c_k$并终止本算法。

4　实验

可间断运行 K-means聚类算法在每次遇到停机时会丢失当前迭代运算的成果, 恢复后以最近完成的迭代结果为基础继续运行, 所以, 该算法间断运行完成所花费的总时间应比一次性完成所花费的时间多一点, 但两者的差别很小。本文的实验为验证这一点而设计。实验在一台普通的 PC机上进行, 该 PC机的 CPU为 Intel Pentium　D 2. 66 GHz, 内存为 1 GB, 运行的操作系统为 Windows XP。在 Visual Studio. NET 2003上用 C++编程实现可间断运行的 K-means聚类算法。实现的程序针对三个不同的数据集分别以一次性完成和间断运行完成两种方式完成聚类任务, 运行的情况如表 1~3所示。表 1~3所展现的实验结果完全符合预期。

表 1　在 dataSet1上的运行情况

维数 48	点数 267 898	K= 300
一次性完成	k个聚类中心 $C(c_1, c_2, \cdots, c_k)$	随机指定初值
	花费的时间	6. 33 h
间断运行完成	k个聚类中心 $C(c_1, c_2, \cdots, c_k)$	指定与一次性完成一样的初值
	间断的情况	运行过程中强行关机然后恢复运行共 6次
	程序在计算机上运行的总时间	6. 35 h

表 2　在 dataSet2上的运行情况

维数 192	点数 267 898	K= 300
一次性完成	k个聚类中心 $C(c_1, c_2, \cdots, c_k)$	随机指定初值
	花费的时间	24. 11 h
间断运行完成	k个聚类中心 $C(c_1, c_2, \cdots, c_k)$	指定与一次性完成一样的初值
	间断的情况	运行过程中强行关机然后恢复运行 6次, 意外关机然后恢复运行 1次
	程序在计算机上运行的总时间	24. 21 h

(下转第 2069页)

所用的时间; DY-SP表示加入动态速度模式的最优路径查找算法; ST-SP表示固定速度模式下最优路径查找算法; SP表示无速度模式的 A* 最短路径查找算法。

由图 5可以看出,在动态的速度模式下,利用 A* 算法所查找的最短路径汽车行驶所用的时间比加入了速度模式所查找的路径汽车行驶所用的时间长;而用固定的速度模式所查找的路径汽车行驶所用时间与动态速度模式下所查找的路径汽车行驶所用时间相比,前者所用时间较长。

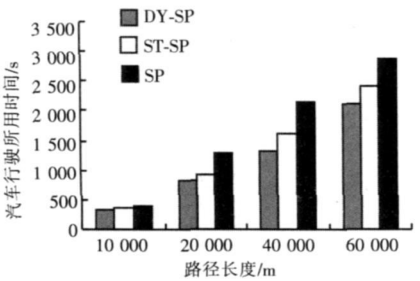

图 5　利用三种算法查找的路径汽车行驶所用时间对比

4　结束语

本文在基于 A* 道路查询算法基础上,加入了速度模式的概念,并且对在道路拥挤致使初始速度模式发生改变的情况下,提出了一种最优路径的查找算法。该算法通过建立速度模式库,对速度模式发生变化的情况进行处理,给出在此情况下的最优路径。这样使对路径查询问题的处理更为全面,更加真实地模拟了道路交通的实际状况。另外,该方法在实际应用中还有若干问题有待解决,如速度模式的有效性以及交通突发时间的影响问题等,都有待于进一步解决。

参考文献:

[1] DIJKSTRA E. A note on two problems in connexion with graphs[J]. Numerische Mathematik, 1959, 1(1): 269-271.

[2] ZHAN F B, NOON C E. Shortest path algorithms: an evaluation using real road networks[J]. Transportation Science, 1998, 31(1): 65-73.

[3] BRINKHOFF T. A framework for generating network-based moving objects[J]. GeoInformatica, 2002, 6(2): 153-180.

[4] SHEKHAR S, YOO J S. Processing in-route nearest neighbor queries: a comparison of alternative approaches[C]//Proc of ACM GIS Conference on Management of Data. New York: ACM Press, 2003: 9-16.

[5] GONZALEZ H, HAN Jiawei, LI Xiaolei. Adaptive fastest path computation on a road network: a traffic mining approach[C]//Proc of the 33rd International Conference on Very Large Data Bases. Austria: VLDB Press, 2007: 23-28.

[6] KANOULAS E, DU Y, XIA T, *et al*. Finding fastest paths on a road network with speed patterns[C]//Proc of the 22nd International Conference on Data Engineering. Washington DC: IEEE Computer Society Press, 2006: 10.

[7] GONZALEZ H, HAN Jiawei, LI Xiaolei, *et al*. Warehousing and analyzing ofmassive RFID data sets[C]//Proc of the 22nd International Conference on Data Engineering. Washington DC: IEEE Computer Society, 2006: 83.

[8] SALTENIS S, JENSEN C S, LEUTENEGGER S T, *et al*. Indexing the positions of continuously moving objects[C]//Proc of ACM SIGMOD International Conference on Management of Data. Dallas: ACM Press, 2000: 331-342.

(上接第 2055页)

表 3　在 dataSet3上的运行情况

维数 768	点数 267 898	K = 300
一次性完成	k个聚类中心 $C(c_1, c_2, \cdots, c_k)$	随机指定初值
	花费的时间	103.25 h
间断运行完成	k个聚类中心 $C(c_1, c_2, \cdots, c_k)$	指定与一次性完成一样的初值
	间断的情况	运行过程中强行关机然后恢复运行3次,意外关机然后恢复运行 4次
	程序在计算机上运行的总时间	103.79 h

表 4所展现的是可间断运行 K-means聚类算法在一个大数据集上运行的情况。该数据集较大,需要的计算时间很长,超过了普通 PC机承受连续运算的能力,因此没有一次性完成的情况。由于 dataSet1~4是从相同的视频流采集来的,只是数据的维数不同而已,根据 dataSet1~3的运行情况,可以粗略地估计出可间断运行 K-means聚类算法在 dataSet4上运算的时间为 400多个小时。表 4展现的情况与估计大体一致。可间断运行 K-means聚类算法在 dataSet4上最终完成聚类运算,说明它的目标已完全达到。

表 4　在 dataSet4上的运行情况

维数 3072	点数 267 898	K = 300
间断运行完成	k个聚类中心 $C(c_1, c_2, \cdots, c_k)$	随机指定初值
	间断的情况	运行过程中强行关机然后恢复运行12次,意外关机然后恢复运行 8次
	程序在计算机上运行的总时间	427.39 h

5　结束语

本文以聚类算法的研究与应用为背景,分析研究了 K-means聚类算法的问题并将该算法的问题归纳为三个方面。针对第三方面的问题提出了自己的解决方案,即引入事务的恢复机制改进 K-means聚类算法。改进后的算法在实验中的运行情况完全符合预期,改进后的算法在总运行时间 400多个小时的聚类任务中经受了考验,使得在普通 PC机上针对大数据集运用 K-means聚类算法成为可能。

参考文献:

[1] 孙吉贵,刘杰,赵连宇. 聚类算法研究[J]. 软件学报, 2008, 19(1): 48-61.

[2] KANUNGO T, MOUNT D M, NETANYAHU N S, *et al*. A local search approximation algorithm for K-means clustering[J]. Computational Geometry, 2004, 28(2-3): 89-112.

[3] 陈黎飞,姜青山,王声瑞. 基于层次划分的最佳聚类数确定方法[J]. 软件学报, 2008, 19(1): 62-72.

[4] BRADLEY P S, FAYYAD U M. Refining initial points for K-means clustering[C]//Proc of the 15th International Conference on Machine Learning. San Francisco: Morgan Kaufmann, 1998: 91-99.

[5] ALSABTI K, RANKA S, SINGH V. An efficient K-means clustering algorithm[C]//Proc of the 1st Workshop on High Performance Data Mining. 1998.

[6] KANUNGO T, MOUNT D M, NETANYAHU N S, *et al*. An efficient K-means clustering algorithm: analysis and implementation[J]. IEEE Trans on Pattern Analysis and Machine Intelligence, 2002, 24(7): 881-892.