在 MapReduce 和 Streaming 场景下进行带有异常值的 k 中心聚类，可以实现足以媲美顺序算法的精确度

翻译：王瀚橙　　学号：55161125
（2016 级本科生 11 班）

译注：本文基于更大的可组合核集这一 idea，针对带有z个离群值和不带有离群值的k中心问题，提出了在 MapReduce 和 Streaming 两种场景下的近似解法。在翻译过程中，我省去了繁杂的算法空间复杂度数学证明过程，详细介绍了每种框架的算法执行过程和执行效果。全文译文如下：

摘要：基于中心的聚类是数据分析领域的基本原语，但很难处理大规模的数据集。在本文中，我们主要研究时下流行的一种 k 中心问题的变体，该变体给定了某度量空间中的点集合 S 以及一个参数k(其中$k \leq |S|$)，要求在集合 S 中标识出k个子集，并最小化 S 中任意点到距离它最近的中心的最大距离。为了处理含有噪音点的数据，我们引入另一个参数z，当我们计算最大距离时，我们最多可以忽略z个离群点。我们提出了基于核集的二轮 MapReduce 算法，可以解决含离群点和不含离群点两种情况。我们又提出了一遍 Streaming 算法，可以解决含有离群点的情况。我们的算法和那些经典的多项式时间算法的近似率仅差了一个固定的常数项ε，这大大改善了现有的技术。我们的算法非常简单，且可以应对数据集的内在复杂性，该复杂性可以通过度量空间的倍数维度 D 捕获。并且对于 D 比较小的情况，我们的算法可以非常节省空间。我们的理论经过了多达十亿个点的真实和合成数据集的验证，相比现有算法在保证可扩展性的同时，质量更高，运行速度更快。

1 引言

基于中心的聚类是用于数据管理的基本无监督学习原语，在数据库检索，生物信息学，模式识别，网络，设备定位等诸多领域有着广泛的应用。总的来说，基于中心的聚类是根据相似性将一组数据划分成很多簇，每个簇的代表元素被称为中心。关于中心聚类的顺序执行算法的研究有很多。但是爆炸性的数据增长让这些算法望而却步，它们或许在小数据集上有效，但大数据上显得力不从心。因此，为针对处理大数据的典型计算框架（如 MapReduce 和 Streaming）量身定制高效聚类算法至关重要。

在本文中，我们专注于k中心问题，正式定义如下：给定度量空间中的一个点集合S以及一个正整数$k \leq |S|$，找到k个点的集合T，其中$T \subseteq S$，我们将这个集合中的k个点称为k个中心，使得S中的任意一个点到T中最近的中心的最大距离最小（请注意，每个点和离它

最近的中心的关联自然的定义了聚类）。基于中心聚类的“k中心”和需要最小化所有距离之和的“k中值”以及需要最小化组内距离平方和的“k均值”三者最近被证明是数据和图形分析的关键原语，如何在大数据领域解决这些问题引起了学界的广泛关注。

由于k中心问题是一个 NP 问题，所以需要近似解。并且由于这个问题需要考虑全局，所以极其容易受到几个离群点的严重影响。然而，现实生活中，离群点是许多数据集中固有的存在，这些数据可能是收集时人为造成的，也可能是噪音甚至是错误的数据。为了解决这个问题，k中心使用了一种考虑了离群点的公式：当我们计算目标函数时，最多可以舍弃z个点，其中z是用户自己输入的参数。

针对这种大数据的组合优化问题，求解其近似解的一种自然有效的方法是提取输入数据的一个较小的子集，我们称之为核集。这个集合包含了对于全局最优值的近似，将顺序近似算法应用于这个核集从而快速获得大数据的近似最优解。如果构造核集比直接在整个数据集上运行顺序近似算法快得多，这种方案的好处就显现出来了。引文 27 的作者基于比输入集合更小的核集，提供了带有和不带有异常值的k中心的 MapReduce 算法，但是该算法的逼近因子远大于最佳顺序算法的结果，效果不好。在本文中，我们进一步利用了核集的思路，揭示了核集大小和近似结果质量之间的关系，并展示了通过构造更大的核集可以实现更好的近似这一现象。具体权衡可以通过基本度量空间的倍数维度调节，从而获得改进的k中心 MapReduce 和 Streaming 算法。该算法的近似率非常接近最佳顺序算法的结果。此外，作为副产品，我们获得了一种针对异常值的顺序算法，该算法比现有算法快得多。

1.1 相关工作

早在 80 年代，Gonzalez 针对k中心问题开发了一种$O(k\,|S|)$复杂度的二近似顺序算法，在最近的文献中常被称为 GMM 算法。Gonzalez 在该论文中表明，对于固定的$\varepsilon > 0$，在基本度量空间中，除非$P = NP$，否则不可能获得近似因子$2-\varepsilon$。为了解决数据集中的噪声，Charikar 等人引入了z个离群点的k中心问题，允许忽略输入数据中的z个离群点。对于这个问题，他们提供了一种$O(k\,|S|^2 \log|S|)$复杂度的三近似算法。并且他们证明，对于固定的$\varepsilon > 0$，在基本度量空间中，除非$P = NP$，否则不可能获得近似因子$3-\varepsilon$。

随着海量数据的到来，MapReduce 计算模型引起了人们的极大关注。MapReduce 使用一组具有有限内存的处理器并行地处理数据。Ene 等人首先研究了该模型下的k中心问题，他们提供了一种十近似的随机算法。这个成果在之后的引文[27]中被改进成只需要$O\left(\sqrt{k\,|S|}\right)$空间复杂度的四近似算法。对于带有离群点的$k$中心问题，引文[27]中提供了空间复杂度为$O\left(\sqrt{(k+z)\,|S|}\right)$的十三近似 MapReduce 算法。我们从引文[24]注意到现在已经有了针对k中心问题的随机多轮 MapReduce 算法，并且具有大约 2 和 4 的近似度。从理论

上讲，在我们的工作中提出的 MapReduce 算法在回合和空间复杂性方面与引文[24]中的算法相比更具有竞争力，但是任何比较都要取决于更多的细节。

正如上文提到的，引文[27]中的算法基于可组合核集的使用，这是 MapReduce 中非常有用的设计。对于给定的目标函数，核集是从输入中提取的一小部分子集，这种解决方案的代价和整个集合上的最优解的代价相近。可组合性指的是如果从输入的不同子集（划分）中提取核集，他们的组合可作为整个输入的最佳近似解。可组合的核集支持了并行算法的开发，每个处理器分别的计算一个子集的核集，之后将核集组合起来进行最终解的计算。可组合核集被应用于诸如多样性最大化、次模最大化、图形匹配和顶点覆盖。在引文 8 中，作者为d维欧氏空间的k中心问题提供了基于核集的$1+\varepsilon$ 逼近顺序算法，该算法的运行时间是k和$(1/\varepsilon)^2$的指数倍，是d和$|S|$的线性倍。但是该算法的核集构造不但十分复杂，且不容易并行化，因此该算法主要具有理论价值。

处理大数据时的另一种选择是流式处理。在流式处理中，算法使用有限内存的单个处理器，并且允许对输入进行几遍连续传输（理性情况下仅一遍传输）。该模型最初是为外部存储器开发的，它捕获动态生成的并需要实时分析的数据，例如流式数据库管理系统或者社交媒体平台例如推特的趋势检测。在这个模型之下，Charikar 等人为k中心问题开发了需要$\theta(k)$空间复杂度的一遍流式算法。该算法可以确定性地计算 8 近似或者概率性的计算 5.43 近似。之后在引文 28 中实现了确定的$2+\varepsilon$ 近似改进算法，该算法需要$\theta(k\varepsilon^{-1}\log\varepsilon^{-1})$空间。依然是引文 28 中，作者给出了针对离群点的$4+\varepsilon$ 近似流式算法，需要$O\left(\sqrt{kz\varepsilon^{-1}}\right)$空间复杂度。

1.2 我们的贡献

引文 27 中的基于核集的（有无异常值的）k中心问题使用了自助抽样的 GMM 顺序近似算法。首先，在输入的任意划分中确定k个中心（有 z 个离群点的确定$k+z$个中心），之后使用线性逼近算法，最终的结果在这些中心的并集组成的核集上运行得到最终结果。我们的工作受到如下的自然想法的启发：如果第一步的时候从每个划分的子集中选择更多的中心点会怎么样。从直觉上讲，我们应该能获得比仅仅选择k个中心（或者$k+z$个中心）更好的解决方案。事实上，从每个子集中选择更多的中心将产生逐渐接近于在整个输入上运行最佳顺序算法的结果，但这样会占用更多的空间。

本文通过对有无异常值应用上述的启发思想，对空间和精度的权衡进行了全面的描述。我们提出了改进的 MapReduce 和 Streaming 算法，该算法选择较大的可组合的核集来提高联合核集思路的质量。我们根据所需要的近似质量来分析算法的内存需求。这个近似质量由精度参数 ε 和基础度量空间的倍数维度 D 度量。这个 D 是一个将欧几里得空间的维数推广到任意度量空间的参数，与发现好聚类的难度有关。我们注意到，这种参数分析在大数据领域非常重要，在大数据领域为了解决最坏情况引入的畸变可能过于极端，影响了

我们对于实际算法性能的见解。这种参数化分析已经在许多方法中使用，比如多样性最大化，聚类，最近邻搜索，路由，机器学习和图分析等等。

我们的具体成果如下：

• 确定的二轮，$2+\varepsilon$ 近似k中心 MapReduce 算法，空间复杂度$O\left(\sqrt{|S|k}(4/\varepsilon)^D\right)$

• 确定的二轮，$3+\varepsilon$ 近似有 z 个离群点的k中心的 MapReduce 算法，空间复杂度为$O\left(\sqrt{|S|(k+z)}(24/\varepsilon)^D\right)$

• 随机的二轮，$3+\varepsilon$ 近似有 z 个离群点的k中心的 MapReduce 算法，空间复杂度为$O\left(\left(\sqrt{|S|(k+\log|S|)}+z\right)(24/\varepsilon)^D\right)$

• 确定的一轮，$3+\varepsilon$ 近似有 z 个离群点的k中心的 Streaming 算法，空间复杂度为$O\left((k+z)(96/\varepsilon)^D\right)$

使用我们的核集构造法我们还可以得到有离群点的$2+\varepsilon$ 近似k中心 Streaming 算法，但是并不会在最新算法上有所改进，但为了完整起见，我们将在第五节中通过实验比较这两种算法。

我们的算法具有近似保证，比最佳顺序算法多一个ε，并且在最先进的技术的基础上取得实质性的质量改进。并且具有离群值的随机二轮 MapReduce 算法倾向于更小的核集，因此减少了局部存储需求，并且当离群点的数目z（可能由于噪音引入）远远大于簇的数目k的情况下，空间节省将变得非常可观。

虽然我们的算法适用于一般度量空间，但是在常数倍维度 D 的空间上，其空间需求在 MapReduce 设置中与数据集大小是多项式次线性的，在 Streaming 中与数据集大小无关。

此外，我们的 MapReduce 算法中未明确使用 D，因为 D 的值可能难以评估并且预先未知的，我们仅仅在分析时使用了 D。然而在一遍 Streaming 算法中确实使用了 D，但是可以以多跑一遍的代价来忽略 D。

并且有异常值的 MapReduce 算法可以直接顺序实现，这大大提高了现有算法的时间性能，同时基本保证了近似结果的质量。

我们将引文 27 中的 MapReduce 和引文 28 中的 Streaming 算法作为基线，之后在高达 10 亿个点的真实数据和合成数据上验证我们算法的竞争力。实验表明 MapReduce 中使用更大的核集可以实现严格的近似。事实上我们的空间需求的理论界限包含很大的常数因子，但是随着核集的增大，我们的逼近质量改善明显。Streaming 算法在没有离群值的情况下我们的算法和引文 28 相当，但有离群值的时候我们的解决方案质量更高，内存和时间更少。实验表明，Streaming 算法的吞吐量高，MapReduce 算法具有高扩展性。最后，我们证明了顺序执行我们的核集策略的确比运行引文 17 中的最新算法更快更好。

文章的剩余部分组织如下：在第二部分中有许多基本概念，第三节和第四节分别介绍了我们的 MapReduce 和 Streaming 算法，实验部分在第五节中，第六节进行结论总结。

2 前提条件

考虑具有距离函数$d(\cdot,\cdot)$的度量空间 S，对于$u \in S$，以u为中心r为半径的球是距离u最多为r的点的集合。S的倍数维 D 指的是，找到最小的 D，使得所有在半径为r中心为u的球中的点可以被2^D个以适宜点为中心$r/2$为半径的球覆盖。也就是说，对于$0 < \varepsilon \leq 1$，一个半径为r的球可以被$(1/\varepsilon)^D$个半径为εr的球覆盖。具有成倍加倍维数的度量空间的显着示例是欧几里得空间和在温和扩展拓扑中由最短路径距离引起的空间。实际上，我们算法的空间精度折衷仅取决于输入数据集的加倍维数。

接下来，我们定义一个点$s \in S$和一个集合$X \subseteq S$的距离为$d(s,X) = \min_{x \in X} d(s,x)$。现在我们考虑一个数据集$S \subseteq S$和子集$T \subseteq S$，我们定义以 T 为代表的 S 的半径为：

$$r_T(S) = \max_{s \in S} d(s,T).$$

k中心问题就是要找到一个大小为k的子集$T \subseteq S$，使得$r_T(S)$最小。我们定义$r_k^*(S)$为此问题的最优解，此外，T 通过将每个点分配到距离其最近的中心，将 S 分为了k个簇，我们将$r_T(S)$视为簇的半径。

在 1.1 中，我们提到了 GMM 算法，为k中心问题提供了顺序 2 近似算法。这里我们简单介绍一下 GMM 算法：给定一个集合S，GMM 算法通过 k 次迭代构建中心子集 T。算法首先选择任意一个S中的点作为第一个中心，将其添加到子集T中。之后，该算法迭代选择下一个中心作为与T最大距离的点，并将其添加到T中，直到 T 包含 k 个中心为止。此外，一开始可以不设置k的值，让 GMM 不断迭代直到到达我想要的半径为止。实际上，S的半径是迭代次数的非递增函数。在本文中，我们将利用 GMM 的如下属性，该属性在对数据的子集运行算法时会限制其准确性：

【引理一】 若$X \subseteq S$，对于给定的k，若T_X是在集合X上运行 GMM 算法的输出，那么$r_{T_X}(X) < 2 \cdot r_k^*(S)$. 证明略。此外，没有异常值的$k+z$中心问题的最优解的半径显然比有$z$个异常值的最优解的半径小，也就是说：

$$r_{k+z}^*(S) \leq r_{k,z}^*(S). \tag{1}$$

2.1 计算框架

MapReduce 算法按照一系列并行的轮次执行。在一轮中，首先通过将给定的映射函数 mapper 应用于每个单独的键值对，将键值对的多集 X 转换为新的键值对的多集X'，然后通过使用给定的归约函数 reducer 独立地作用于具有相同键值的X'。该模型有两个参数：M_L（每个映射器/归约器可用的本地内存）和M_A（所有映射器/归约器之间的聚合内存）。在我们的算法中，mapper 是简单的常量空间转换，因此内存需求将与 reducer 有关。此外我们注意到，本文介绍的 MapReduce 算法也是在数据库社区中很流行的大规模并行计算（MPC）模型的一种实现和类比分析。

Streaming 是用具有较小工作内存的单个处理器执行的，并且输入是连续的项目流，该项目流通常太大而无法容纳在工作内存中。输入流可以多次通过处理器，Streaming 的关键性能指标是处理器的存储器大小以及通过遍数。

大数据算法的发展是 MapReduce（Streaming）算法的发展，该算法需要尽可能少的轮次（遍数）并需要基本亚线性的局部内存（工作内存）和线性聚合内存。

3 MapReduce 算法的设计

接下来我们将在 3.1 小节中介绍不带有异常值的k中心问题的 MapReduce 解法，在 3.2 小节中介绍带有异常值的k中心问题的 MapReduce 算法。我们基于引言中的可组合核集改进了引文 27 中的算法。我们的算法的创新点在于合理地利用了更大的核集达到近似于最著名的顺序算法所具有的近似质量。并且我们在分析过程中说明：我们的核集大小与基本度量空间的倍增维度有关，虽然算法执行过程中并不需要具体的倍增维度，但我们说明了对于有界倍增维度的空间，核集的大小保持在一个较小的尺寸。

3.1 k中心问题的 MapReduce 算法

考虑k中心问题的一个实例S，以及一个合适的用于衡量近似度的参数$\varepsilon \in (0,1]$，MapReduce 算法分两轮来进行处理。在第一轮中，S 被均分成ℓ个子集S_i，其中$1 \le i \le \ell$。在每个S_i上并行运行 GMM 算法，我们用T_i^j表示S_i运行 GMM 算法j次迭代获得的j个中心。我们用$r_{T_i^k}(S_i)$表示以前k个点为中心的S_i的半径。我们不断地运行 GMM 算法直到运行到第$\tau_i \ge k$次迭代，使得$r_{T_i^{\tau_i}}(S_i) \le \varepsilon/2 \cdot r_{T_i^k}(S_i)$ 并且我们定义核集$T_i = T_i^{\tau_i}$。第二轮中，核集的并集$T = \bigcup_{i=1}^{\ell} T_i$被汇总为一个 reducer 之后在$T$上运行 GMM 获得$k$个中心。接下来，作者证明了这样求得的$k$个中心是$k$中心问题的优良答案，此处限于篇幅暂不详细说明。

【推论一】我们的二轮$2+\varepsilon$ 近似k中心 MapReduce 算法需要$M_L = O\left(\sqrt{|S|k}(4/\varepsilon)^D\right)$对于确定的$\varepsilon$和 D，本地内存的上界为$M_L = O\left(\sqrt{|S|k}\right)$。

3.2 有z个离群值的k中心问题的 MapReduce 算法

和上文一样，考虑k中心问题的一个实例S，以及一个合适的用于衡量近似度的参数$\hat{\varepsilon} \in (0,1]$，我们提出的 MapReduce 算法分两轮来进行处理。在第一轮中，S 被均分成ℓ个子集S_i，其中$1 \le i \le \ell$。在每个S_i上并行运行 GMM 算法，我们用T_i^j表示S_i运行 GMM 算法j次迭代获得的j个中心。我们不断地运行 GMM 算法直到运行到第$\tau_i \ge k+z$次迭代，使得$r_{T_i^{\tau_i}}(S_i) \le \hat{\varepsilon}/2 \cdot r_{T_i^{k+z}}(S_i)$ 并且我们定义核集$T_i = T_i^{\tau_i}$。像上文那样，对于每个$s \in S_i$，我

们定义一个代理函数$p(s)$，将s映射到T_i中离它最近的点。但是和之前不同，我们为T_i中的每一个t定义了一个权重$w_t \geq 1$表示映射到这个点的s的数目。

在第二轮中，我们将加权后的核集$T = \bigcup_{i=1}^{\ell} T_i$汇总为一个 reducer，在介绍第二轮的具体操作之前，我们先引入一个名为 Outliers Cluster 的顺序算法，用于解决含有z个离群点的k中心问题。未加权的算法最早出现于引文 17，之后加权的算法在引文 27 中提出。

该算法的伪代码如下：

Algorithm 1: OUTLIERSCLUSTER$(T, k, r, \hat{\varepsilon})$

$T' \leftarrow T$
$X \leftarrow \emptyset$
while $((|X| < k)$ **and** $(T' \neq \emptyset))$ **do**
  for $(t \in T)$ **do** $B_t \leftarrow$
    $\{v : v \in T' \wedge d(v,t) \leq (1+2\hat{\varepsilon}) \cdot r\}$
  $x \leftarrow \arg\max_{t \in T} \sum_{v \in B_t} w_v$
  $X \leftarrow X \cup \{x\}$
  $E_x \leftarrow \{v : v \in T' \wedge d(v,x) \leq (3+4\hat{\varepsilon}) \cdot r\}$
  $T' \leftarrow T' \setminus E_x$
return X, T'

图 1 Outliers Cluster 算法伪代码

Outliers Cluster 算法返回两个集合X和$T' \subseteq T$，其中X最多是k个中心，T'是为未被覆盖到的点。算法开始时，T'被赋值为T，之后在最多k次迭代中构建X。在每一次迭代中，我们找出那个可以使在以其为中心，以$(1+2\hat{\varepsilon}) \cdot r$为半径的球中所有的未被覆盖到的点的加权最大的那个$x$。注意，我们没有要求$x$必须是未被覆盖的点。之后，我们将距离 x 最远$(3+4\hat{\varepsilon}) \cdot r$的未被覆盖到的点移出$T'$。当$|X| = k$或者$T' = \emptyset$时算法停止，最终在$T'$中的未被覆盖到的点是距离$X$超过$(3+4\hat{\varepsilon}) \cdot r$的点。

让我们再回到 MapReduce 算法的第二轮，reducer 会多次运行 Outliers Cluster 算法以评估其中的参数r的值，通过寻找一个r_{min}使得T'之中仅仅有z个异常值。更确切的说，reducer 通过对所有的$O(|T|^2)$种可能使用二进制搜索，进行以$(1+\delta)$为步长的几何搜索获得距离真正的r_{min}公差为$(1+\delta)$的$\tilde{r}_{min}$，其中$\delta = \hat{\varepsilon}/(3+4\hat{\varepsilon})$。为了避免存储所有$O(|T|^2)$的$r$的值，我们使用引文 31 中的 Streaming 算法使用线性空间确定每次的r的值。Outliers Cluster 算法的输出作为 MapReduce 算法的最终结果。该算法的具体证明暂不详述。

【推论二】我们的二轮$3+\varepsilon$ 近似的带有z个离群值的k中心 MapReduce 算法需要$M_L = O\left(\sqrt{|S|(k+z)}(24/\varepsilon)^D\right)$的本地内存空间以及聚合内存空间。对于确定的$\varepsilon$和 D，本地内存的上界为$M_L = O\left(\sqrt{|S|(k+z)}\right)$。

改进后的顺序算法：为了简单起见，我们假设$\ell = 1$，简单分析表明，我们的二轮$3 + \varepsilon$ 近似的带有z个离群值的k中心 MapReduce 算法时间复杂度为$O(|S||T| + k|T|^2 \log|T|)$其中，核集的大小为 $|T| = (k + z)(24/\varepsilon)^D$。我们使用了微不足道的近似度作为代价，大大改善了引文 17 中的$O(k|S|^2 \log|S|)$的时间复杂度。

3.2.1 通过随机化提高空间效率

对于非常嘈杂的数据集，离群点的数目z的值可能会远远大于k的值，使得核集变成$\sqrt{|S|z}$，由于实际值可能很大，过大的本地内存的过慢的三近似顺序算法将成为整个算法的瓶颈。在本小节中，我们表明可以通过在第一轮中简单地随机划分点集来显着地改善此缺点，这仅是对结果空间的概率近似保证而非确定性保证。也就是说对于常数$c \geq 1$，这个事件将以$p \geq 1 - 1/|S|^c$的概率发生。

我们算法的工作方式如下：在第一轮中，我们将 S 被到ℓ个子集S_i，其中$1 \leq i \leq \ell$，每个s分配给一个统一且独立于其他点选择的随机子集。我们设置$z' = 6\big((z/l) + \log_2|S|\big)$，显然对于非常大的$z$和$\ell$，$z' \ll z$。在每个$S_i$上并行运行 GMM 算法，我们用$T_i^j$表示$S_i$运行 GMM 算法$j$次迭代获得的$j$个中心。我们不断地运行 GMM 算法直到运行到第$\tau_i \geq k + z'$次迭代，使得：

$$r_{T_i^{\tau_i}}(S_i) \leq \hat{\varepsilon}/2 \cdot r_{T_i^{k+z'}}(S_i)$$

我们定义核集$T_i = T_i^{\tau_i}$，像上文那样，对于每个$s \in S_i$，我们定义一个代理函数$p(s)$，将s映射到T_i中离它最近的点，之后的第二轮处理过程就和上文一样了。

本算法的空间分析限于篇幅暂不详述。

【推论三】我们的二轮$3 + \varepsilon$ 近似的带有z个离群值的k中心 MapReduce 算法需要 $M_L = O\left(\left(\sqrt{|S|(k + \log|S|)} + z\right)(24/\varepsilon)^D\right)$的本地内存空间以及聚合内存空间。对于确定的$\varepsilon$和 D，本地内存的上界为$M_L = O\left(\left(\sqrt{|S|(k + \log|S|)} + z\right)\right)$。

对于较大的z值，上面的算法比确定性版本节约空间显著。

备注：由于 GMM 算法的增量性质，我们针对 k 中心问题的基于核集的 MapReduce 算法（无论有无异常值）都无需知道基础度量空间的倍数维 D 即可达到所要求的性能范围。这是一个非常理想的属性，因为通常 D 可能事先未知。此外，如果 D 是已知的，通过设置ℓ为$\Theta\left(\sqrt{(c/\varepsilon)^D}\right)$，将会节约$\sqrt{(c/\varepsilon)^D}$的本地内存空间。其中没有异常值的$k$中心问题$c = 4$，对于有$z$个异常值的$k$中心问题$c = 24$。

4 对于有 z 个异常值的 k 中心问题的 Streaming 算法的设计

正如引言中所说，Streaming 算法我们仅仅介绍有z个离群值的k中心问题的解法。考虑k中心问题的一个实例S，以及一个合适的用于衡量近似度的参数$\hat{\varepsilon} \in (0,1]$。假设已知 S 属于已知倍增维数的空间。我们的 Streaming 算法依然使用基于可组合核集的方法，具体来说，S 一次流过，我们选择合适的加权核集 T 并将其存储在工作存储器中。最后像 3.2 小节中第二轮所说，通过在 T 上多次运行 Outliers Cluster 算法确定最后的中心集合。下面我们将重点介绍核集的构建。

我们的核集算法计算出一个包含$\tau \geq k + z$个中心的核集，这个核集是不含有离群值的 S 集合的τ中心问题的最优核集。τ的具体值将由$\hat{\varepsilon}$和D决定，具体决定方式一会再说。核集的具体求法和 MapReduce 的不同在于我们无法利用 GMM 的增量性质，因为现如今还没有 Streaming 中的 GMM 实现。因此我们使用了 Charikar 等人的新型加权变量算法：

给定点流 S 以及目标中心数τ，算法针对所有处理过的点维持一个加权后的核集，并记录$r_\tau^*(S)$的下界ϕ。我们用$\tau + 1$个点初始化核集，并将每个点的权值赋初值为 1，ϕ被初始化为 T 的点之间最小距离的一半。为了方便分析，我们定义一个映射关系$p: S \to T$，该映射关系不用存储，并且初始化 T 中的每个点为到其自身。之后不断流式处理 S 中剩余的点，在处理过程中需要保持下面的约束：

(a) T contains at most τ centers.
(b) $\forall t_1, t_2 \in T$ we have $d(t_1, t_2) > 4\phi$
(c) $\forall s \in S$ processed so far, $d(s, p(s)) \leq 8\phi$.
(d) $\forall t \in T$, $w_t = |\{s \in S \text{ processed so far}: \ p(s) = t\}|$.
(e) $\phi \leq r_\tau^*(S)$.

图 2 Charikar 算法约束条件

有两条规则用于处理 S 中的每一个点：其一为**更新规则**，需要检查输入流中的每一个点是否满足$d(s,T) \leq 8\phi$. 如果满足这种情况，就找到s对应的$t \in T$，并且对应权重自加一，且定义$p(s) = t$。如果$d(s,T) > 8\phi$，则将s加入到核集之中，并将s的权值置为 1，并定义$p(s) = s$。然而这样会违反上面的约束规则中的约束条件(a)，将会调用**合并规则**，直到再次满足约束条件(a)为止。第一次调用合并规则时，我们将ϕ设置为2ϕ，这可能会违反约束条件(b)。对于这种情况，我们对于每一对不满足约束条件(b)的点u和点v，我们丢弃点u，之后将点v的权重更新为u和v原有权重之和。从概念上，将原来映射到点u的点改映射到点v之上。算法的证明限于篇幅暂不详述。

5 实验

为了证明我们的方法的实际性能，我们设计了一套实验，其目标如下：（a）评估我们的 MapReduce 和 Streaming 算法中核集大小对解决方案质量的影响，并将其与带有和不带有

离群值的 k 中心问题的最新算法做对比，详见 5.1 和 5.2 小节。（b）评估我们的 MapReduce 算法的可扩展性，详见 5.3 小节。（c）证明对于没有异常值的 MapReduce 算法我们提出了一种更快的针对该问题的顺序算法详见 5.4 小节。

实验设置：我们的实验在由 16 台计算机组成的群集上运行。每台计算机均配备 18GB RAM 和 4 核因特尔 i7 处理器，并通过 10 兆以太网连接。我们使用 Spark 来实现 MapReduce 算法，对于 Streaming 采用了顺序仿真。我们使用了在引文 27 中使用过的两个低维现实生活中的数据集上运行我们的算法，以便于与该工作进行比较，并在一个高维的数据集上对我们的维度敏感策略进行了压力测试。第一个数据集 **Higgs** 包含了 1100 万个用于训练高能物理实验学习算法的点。第二个数据集 **Power** 包含 2,075,259 个点，这些点是四年中房屋中的电力消耗的度量。**Higgs** 数据集具有 28 个属性，其中 7 个是其他 21 个属性的函数。在引文 27 中，仅使用了那 7 个派生属性，为了与之比较，我们也这样做。**Power** 数据集具有 7 个数字属性，我们忽略了其中两个非数字属性。第三个较高维度的数据集是使用了 50 个维度的 word2vec 模型从英语维基百科（2017 年 12 月）的转储中获得的。这个数据集（我们称为 **Wiki**）包含 5,512,693 个向量。为了测试我们算法的可扩展性，我们还生成了 **Higgs**、**Power** 和 **Wiki** 的人工实例，详见 5.3 小节的详细信息。对于所有的数据集我们使用欧式距离，所有的数值均是运行十次取平均值，并且结合 95%的置信区间记录在图表中。我们使用近似率表示结果的质量，我们将我们的结果和在相同数据集相同参数下的所有实验中找到的最佳半径的比例作为近似率。毕竟问题的复杂度使得我们没法获得最优解。

6 结论

我们提出了基于可扩展的核集构造了 k 中心问题（有无异常值）的 MapReduce 和 Streaming 算法。这些算法产生了受基础空间的倍数维 D 约束的精度权衡。并且我们的分析有实验的支撑。

未来的研究方向包括进一步改善 MapReduce 算法的空间需求，开发一种可以忽略 D 的一遍流式算法，以及把我们的方法扩展应用到其他的基于中心的聚类问题之中。

参考文献

[1] Twitter Blog: New Tweets per Second Record, and How! https://blog.twitter.com/engineering/en_us/a/2013/ new-tweets-per-second-record-and-how.html.

[2] UCI higgs dataset. https://archive.ics.uci.edu/ml/datasets/HIGGS.

[3] UCI power dataset. https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption.

[4] P. Agarwal, S. Har-Peled, and K. Varadarajan. Approximating Extent Measures of Points. *Journal of the ACM*, 51(4):606–635, 2004.

[5] S. Aghamolaei, M. Farhadi, and H. Zarrabi-Zadeh. Diversity Maximization via Composable Coresets. In *Proc. CCCG*, 2015.

[6] A. Assadi and S. Khanna. Randomized Composable Coresets for Matching and Vertex Cover. In *Proc. ACM SPAA*, pages 3–12, 2017.

[7] P. Awasthi and M. Balcan. Center based clustering: A foundational perspective. In *Handbook of cluster analysis*. CRC Press, 2015.

[8] M. Badoiu, S. Har-Peled, and P. Indyk. Approximate Clustering via Core-sets. In *Proc. ACM STOC*, pages 250–257, 2002.

[9] P. Beame, P. Koutris, and D. Suciu. Communication Steps for Parallel Query Processing. In *Proc. ACM PODS*, pages 273–284, 2013.

[10] M. Ceccarello, C. Fantozzi, A. Pietracaprina, G. Pucci, and F. Vandin. Clustering Uncertain Graphs. *PVLDB*, 11(4):472–484, 2017.

[11] M. Ceccarello, A. Pietracaprina, and G. Pucci. Fast Coreset-based Diversity Maximization under Matroid Constraints. In *Proc. ACM WSDM*, pages 81–89, 2018.

[12] M. Ceccarello, A. Pietracaprina, and G. Pucci. Solving *k*-center Clustering (with Outliers) in MapReduce and Streaming, almost as Accurately as Sequentially. *CoRR*, abs/1802.09205, 2018.

[13] M. Ceccarello, A. Pietracaprina, G. Pucci, and E. Upfal. Space and Time Efficient Parallel Graph Decomposition, Clustering, and Diameter Approximation. In *Proc. ACM SPAA*, pages 182–191, 2015.

[14] M. Ceccarello, A. Pietracaprina, G. Pucci, and E. Upfal. A Practical Parallel Algorithm for Diameter Approximation of Massive Weighted Graphs. In *Proc. IEEE IPDPS*, 2016.

[15] M. Ceccarello, A. Pietracaprina, G. Pucci, and E. Upfal. MapReduce and Streaming Algorithms for Diversity Maximization in Metric Spaces of Bounded Doubling Dimension. *PVLDB*, 10(5):469–480, 2017.

[16] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental Clustering and Dynamic Information Retrieval. *SIAM J. on Computing*, 33(6):1417–1440, 2004.

[17] M. Charikar, S. Khuller, D. Mount, and G. Narasimhan. Algorithms for Facility Location Problems with Outliers. *In Proc. ACM-SIAM SODA*, pages 642–651, 2001.

[18] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *J. Artif. Intell. Res.*, 16:321–357, 2002.

[19] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[20] A. Ene, S. Im, and B. Moseley. Fast Clustering Using MapReduce. In *Proc. ACM KDD*, pages 681–689, 2011.

[21] T. Gonzalez. Clustering to Minimize the Maximum Intercluster Distance . *Theoretical Computer Science*, 38:293–306, 1985.

[22] C. Hennig, M. Meila, F. Murtagh, and R. Rocci. *Handbook of cluster analysis*. CRC Press, 2015.

[23] M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on Data Streams. In *Proc. DIMACS*, pages 107–118, 1998.

[24] S. Im and B. Moseley. Brief Announcement: Fast and Better Distributed MapReduce Algorithms for k-Center Clustering. In *Proc. ACM SPAA*, pages 65–67, 2015.

[25] P. Indyk, S. Mahabadi, M. Mahdian, and V. Mirrokni. Composable Core-sets for Diversity and Coverage Maximization. In *Proc. ACM PODS*, pages 100–108, 2014.

[26] J. Leskovec, A. Rajaraman, and J. Ullman. *Mining of Massive Datasets, 2nd Ed*. Cambridge University Press, 2014.

[27] G. Malkomes, M. Kusner, W. Chen, K. Weinberger, and B. Moseley. Fast Distributed k-Center Clustering with Outliers on Massive Data. In *Proc. NIPS*, pages 1063–1071, 2015.

[28] R. McCutchen and S. Khuller. Streaming Algorithms for k-Center Clustering with Outliers and with Anonymity. In *Proc. APPROX-RANDOM*, pages 165–178, 2008.

[29] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed Representations of Words and Phrases and their Compositionality. In *Proc. NIPS*, pages 3111–3119, 2013.

[30] M. Mitzemacher and E. Upfal. *Probability and Computing*. Cambridge University Press, 2nd edition, 2017.

[31] J. Munro and M. Paterson. Selection and Sorting with Limited Storage. *Theor. Comput. Sci.*, 12:315–323, 1980.

[32] A. Pietracaprina, G. Pucci, M. Riondato, F. Silvestri, and E. Upfal. Space-Round Tradeoffs for MapReduce Computations. In *Proc. ACM ICS*, pages 235–244, 2012.

[33] M. Z. V. Mirrokni. Randomized Composable Core-sets for Distributed Submodular Maximization. In *Proc. ACM STOC*, pages 153–162, 2015.

[34] M. Zaharia, M. Chowdhury, M. Franklin, S. Shenker, and I. Stoica. Spark: Cluster Computing with Working Sets. In *Proc. USENIX HotCloud*, 2010.