Efficient Distributed Density Peaks for Clustering Large Data Sets in MapReduce

Yanfeng Zhang, Shimin Chen, and Ge Yu, *Member, IEEE*

Abstract—Density Peaks (DP) is a recently proposed clustering algorithm that has distinctive advantages over existing clustering algorithms. It has already been used in a wide range of applications. However, DP requires computing the distance between every pair of input points, therefore incurring quadratic computation overhead, which is prohibitive for large data sets. In this paper, we study efficient distributed algorithms for DP. We first show that a naïve MapReduce solution (Basic-DDP) has high communication and computation overhead. Then, we propose LSH-DDP, an approximate algorithm that exploits Locality Sensitive Hashing for partitioning data, performs local computation, and aggregates local results to approximate the final results. We address several challenges in employing LSH for DP. We leverage the characteristics of DP to deal with the fact that some of the result values cannot be directly approximated in local partitions. We present formal analysis of LSH-DDP, and show that the approximation quality and the runtime can be controlled by tuning the parameters of LSH-DDP. Experimental results on both a local cluster and EC2 show that LSH-DDP achieves a factor of 1.7–70x speedup over the naïve Basic-DDP and 2x speedup over the state-of-the-art EDDPC approach, while returning comparable cluster results. Compared to the popular K-means clustering, LSH-DDP also has comparable or better performance. Furthermore, LSH-DDP could achieve even higher efficiency with a lower accuracy requirement.

Index Terms—Density peaks, distributed clustering, MapReduce

1 Introduction

Clustering is a common technique widely used in many fields, including data mining, machine learning, information retrieval, image processing, and bioinformatics. *Density Peaks* [1] (*DP*) is a new clustering algorithm proposed in 2014.

Given a set of points, DP computes two metrics for every point p: (i) ρ, the local density, which is the number of points within a specified distance from p; and (ii) δ, the minimum distance from p to other points with higher densities. It is observed that the center of a cluster sees the highest local density among its neighbor points, and has a relatively large distance from other points with higher densities. Therefore, cluster centers can be determined by identifying the points with both high ρ and high δ.

While DP is attractive for its effectiveness and its simplicity, the application of DP is limited by its computational cost. In order to obtain ρ and δ, DP computes the distance between every pair of points. That is, given N points in the input data set, DP's computational cost is $O(N^2)$. As a result, it can be very time consuming to perform DP for large data sets.

In this paper, we study efficient distributed algorithms for DP so that this promising clustering algorithm can be more broadly used. In particular, we design distributed DP algorithms in MapReduce, which is one of the most popular big data processing paradigms today.

Why DP Clustering is Promising. Compared with previous clustering algorithms, DP has the following four advantages.

- First, DP does not require a priori knowledge about the point distribution. In many well-known algorithms, such knowledge is important for choosing good algorithm parameters (e.g., the number of clusters in K-means [2], ε and $minPts$ in DBSCAN [3]). In comparison, the clustering results of DP have been shown to be robust against the initial choice of algorithm parameters.
- Second, DP supports arbitrarily shaped clusters. Its effectiveness does not rely on the distribution of the data. This is in contrast to K-means and related algorithms which assume the clusters are "balls" in space.
- Third, DP is deterministic. It always computes consistent cluster results, while many clustering algorithms (e.g., K-means and EM clustering [4]) may converge to different local minimums with different initial iterative states.
- Last but not least, (ρ, δ) provides a two dimensional representation of the input point data, which can be in very high dimensions. It is straightforward to visualize (ρ, δ) in a 2D decision graph. From the graph, users can gain new insights into the data distribution and intuitively determine cluster centers.

Due to its effectiveness and novelty, DP algorithm is originally published in Science Magazine [1] in June, 2014. In the past two years since its publication, DP has already been employed in a wide range of applications, such as neuroscience [5], geoscience and remote sensing [6], molecular biology [7], computational biophysics [8], image processing [9], time series mining [10], and computer vision [11].

- *Y. Zhang and G. Yu are with the School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China. E-mail: zhangyf@cc.neu.edu.cn, yuge@mail.neu.edu.cn.*
- *S. Chen is with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100000, China. E-mail: chensm@ict.ac.cn.*

Manuscript received 14 Jan. 2016; revised 7 Aug. 2016; accepted 10 Sept. 2016. Date of publication 14 Sept. 2016; date of current version 2 Nov. 2016. Recommended for acceptance by S. Babu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2016.2609423

Authorized licensed use limited to: Northeastern University. Downloaded on April 06,2025 at 03:23:19 UTC from IEEE Xplore. Restrictions apply.
1041-4347 © 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

Challenges for Distributed DP. In a baseline implementation (Basic-DDP, Basic Distributed DP), we compute ρ and δ values in two subsequent MapReduce jobs. The two jobs have similar computation procedures: The Map and the shuffling stages are mainly used to send and prepare the input data, while the Reduce stage performs the actual computation of ρ and δ, respectively. However, the algorithm has to shuffle every point to every other points and compute distances of all pairs of points, incurring quadratic computation and communication cost. Such cost will be prohibitive for large data sets that have millions of points, which are becoming more and more common in the big data era.

We consider approximate algorithms in order to reduce the computation and communication cost of distributed DP. We observe that DP takes advantage of the local characteristics (such as local density) of the data points for clustering. Therefore, it is natural to employ Locality-Sensitive Hashing (LSH) [12] to partition the input data so that closer points are more likely to be assigned to the same partition. Typically, an LSH-based algorithm performs local computation within each partition, and then aggregates the local results from all partitions to obtain the final approximate results.

There are several challenges in employing LSH for DP. The first challenge is the computation of δ. While the local density ρ is a local property, δ is the the minimum distance to other points with higher ρ. Given a point p, it is possible that other points with higher ρ are far away from p and thus do not reside in p's local partition. The second challenge is to provide guarantees for approximation accuracy of δ and ρ. It would be nice if LSH parameters such as the number of hash functions and the number of local partitions can be derived from the approximation accuracy target specified by the user. Finally, the LSH parameters may also impact the runtime of the solution. Therefore, it is important to study the tradeoff between approximation quality and efficiency.

Our Solution: LSH-DDP. To address the above challenges, we propose an approximate algorithm for DP, called LSH-DDP (LSH based Distributed DP). Specifically, we exploit the fact that cluster centers have both high ρ and high δ. Therefore, given a point p, if we cannot find another point with higher ρ in the local partition, then we will consider p as a candidate cluster center. Moreover, we analyze LSH-DDP and prove the approximation accuracy guarantees for ρ and δ. Based on this analysis, we derive the relationship between LSH parameters and the expected approximation quality. Finally, we evaluate the accuracy and performance of LSH-DDP by comparing LSH-DDP with Basic-DDP using real-world data sets with up to 11.6 million data points. Experimental results show that compared to Basic-DDP, LSH-DDP obtains very similar clustering results, while achieving up to 70× speedup.

Contributions. The contributions of the paper are threefold: First, we propose *LSH-DDP*, an efficient distributed algorithm that approximates ρ and δ values in the DP algorithm. Second, we present formal analysis of LSH-DDP. Given a specific result quality requirement, users can tune the parameters to balance between effectiveness and efficiency. Finally, We conduct extensive experiments on real data sets. Experimental results demonstrate that LSH-DDP achieves a factor of 1.7–70× speedup over the naïve Basic-DDP and 2× speedup over the state-of-the-art EDDPC approach, while returning comparable cluster results. Compared to the widely used K-means clustering, LSH-DDP has comparable or better efficiency.

The remainder of the paper is organized as follows. Section 2 describes background on DP and MapReduce. Section 3 introduces the basic MapReduce implementation of DP as baseline. Section 4 proposes and analyzes our LSH-based approximate solution. Section 5 discusses parameter tuning. Section 6 reports the experimental results. Section 7 discusses related work and Section 8 concludes the paper.

2 Density Peaks Clustering Preliminaries

In this section, we review the standard DP algorithm. Density Peaks Cluster [1] is a novel clustering algorithm recently proposed by Rodriguez and Laio. The algorithm is based on two observations: (i) cluster centers are often surrounded by neighbors with lower local densities, and (ii) they are at a relatively large distance from any points with higher local densities. Correspondingly, DP computes two metrics for every data point: (i) its *local density* ρ and (ii) its distance δ from other points with higher density. DP uses the two metrics to locate density peaks, which are the cluster centers.

The local density ρ_i of data point i is computed as

$$\rho_i = \sum_j \chi(d_{ij} - d_c) \tag{1}$$

where $\chi(x) = 1$ if $x < 0$ and $\chi(x) = 0$ otherwise, and d_c is called the cutoff distance. That is, ρ_i is equal to the number of data points within the cutoff distance d_c.

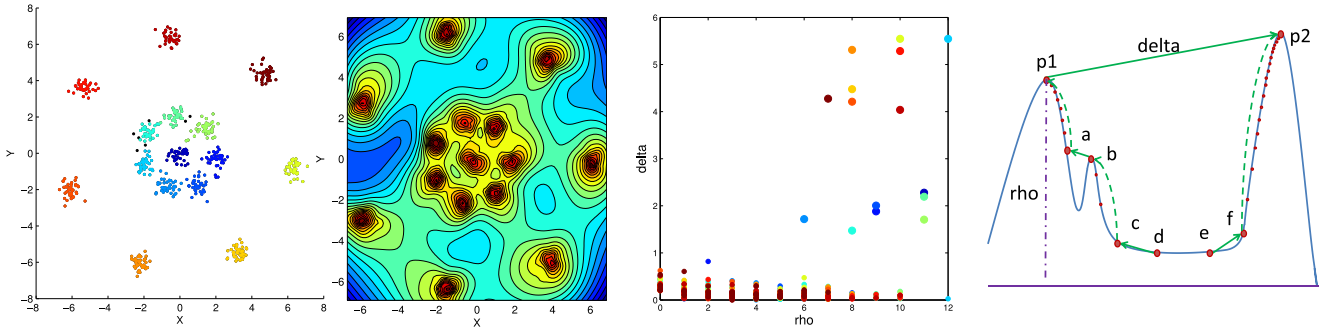
The δ_i distance of data point i is computed as

$$\delta_i = \min_{j|\rho_j > \rho_i} (d_{ij}). \tag{2}$$

It is the minimum distance from point i to any other point whose local density is higher than that of point i. Suppose $j = \arg\min_{j|\rho_j > \rho_i}(d_{ij})$. We say that point i is *assigned* to point j, and point j is referred to as the *upslope point* of point i. If point i has the highest density among all data points, i.e., $i = \arg\max_t \rho_t$, then we set $\delta_i = \max_j(d_{ij})$. This point is called the *absolute density peak*.

Fig. 1 illustrates the process of DP clustering through a concrete example. Fig. 1a shows the distribution of a set of data points. Fig. 1b depicts the corresponding density contour view based on the local density ρ of each point. The warmer the color, the higher the density. Clearly, the peaks of the density mountains (a.k.a. density peaks) correspond to the cluster centers. Then we compute δ. For a normal point i on the slope of a mountain, the closest point that has higher density than i is the next upslope point on the same mountain. This holds for all the points except the density peaks, who will be assigned to points on other higher mountains. This process forms an assignment chain as shown in Fig. 1d, where the height of each point indicates its density ρ. Therefore, *the density peaks are distinguished from other points as they have the* ***highest*** *local density ρ and a* ***large*** *δ*. A point i is depicted on a decision graph as shown in Fig. 1c by using (ρ_i, δ_i) as its *x-y* coordinate. Then the *density peaks* can be identified as outliers in the top right region of the decision graph. Given the selected density peaks

Authorized licensed use limited to: Northeastern University. Downloaded on April 06,2025 at 03:23:19 UTC from IEEE Xplore. Restrictions apply.



(a) Plane view (b) Contour view (c) Decision graph (d) Assignment chain

Fig. 1. Illustrative figures for DP algorithm.

(cluster centers), it is straightforward to follow the assignment chain of a point to determine the density peak and the corresponding cluster that it belongs to.

DP requires the computation of pair-wise distances. A sequential implementation can be improved with the following techniques: (1) For computing ρ, we can employ the triangle inequality to filter unnecessary distance computations. (2) For computing δ, we can first sort the points according to descending ρ values. To compute δ_i, we only need to consider i's distance to the points ahead of point i. Note that these techniques are orthogonal to our proposed techniques and can be easily employed in the sub-tasks of the distributed computation in this paper.

3 Baseline Method

In this section, we describe a basic MapReduce implementation of distributed DP, *Basic-DDP*. We analyze its cost and then discuss improvement opportunities.

3.1 Basic Stategy

In the following, we describe the four steps of Basic-DDP: a preprocessing step for choosing d_c, two key steps for computing ρ and δ values, and the final step for cluster assignment. Table 1 lists the notations used in this paper.

Preprocessing Step: Choosing d_c. The cutoff distance d_c is a key parameter in DP. d_c specifies the meaning of *local* in the computation of the local density ρ in Equation (1). While the DP paper [1] shows that varying d_c (by a factor of 20) produces mutually consistent results, we still need to choose a reasonable d_c without a priori knowledge of the input data. As a rule of thumb, one can choose d_c so that the average number of neighbors is around 1-2 percent of the total number of points in the data set [1]. Suppose the distances between all pairs of points $D_{SEQ} = \{d_{12}, d_{13}, \ldots, d_{21}, \ldots\}$ are known, the 1 or 2 percent position of the ascending ordered set $Ord_a(D_{SEQ})$ can be approximately seen as d_c[1]. Considering that distributed sorting is an expensive task, we rely on sampling (where Reservoir Sampling [13] is used to retrieve a set of sample points) and run a preprocessing MapReduce job to estimate a reasonable d_c

Step 1: Computing ρ. As shown in Equation (1), the computation of ρ requires to know the pairwise distance between all pairs of points. Basic-DDP employs *blocking* technique for pairwise distance computation to save the shuffle cost. The points set S is partitioned into n disjoint subsets, i.e., $S = \cup_{1 \le k \le n} S_k$, where $S_k \cap S_l = \emptyset (\forall k \neq l)$ and $\mathbb{P}_k$ contains the point ids of S_k. The block partitioning is performed by the `map()` function. Since only the upper triangular of the symmetric distance matrix is needed, it sends each subset S_k only to $\{S_l | k \le l \le n\}$ rather than all the subsets. The `reduce()` function is then applied to each pair of subsets (S_k, S_l), where $k < l \le n$, or the diagonal subsets S_k. Based on distance computation, for (S_k, S_l), `reduce()` outputs two sets $\Omega_k^l = \{\rho_i^l | \forall i \in \mathbb{P}_k\}$ where $\rho_i^l = \sum_{j \in \mathbb{P}_l} \chi(d_{ij} - d_c)$ and $\Omega_l^k = \{\rho_i^k | \forall i \in \mathbb{P}_l\}$ where $\rho_i^k = \sum_{j \in \mathbb{P}_k} \chi(d_{ij} - d_c)$. Similarly, for the diagonal subsets S_k, it outputs $\{\rho_i^k | \forall i \in \mathbb{P}_k\}$ where $\rho_i^k = \sum_{j \in \mathbb{P}_k} \chi(d_{ij} - d_c)$. Finally, Basic-DDP runs another MapReduce job to combine the results of ρ_i^l for all $1 \le l \le n$, i.e., $\rho_i = \sum_{l=1}^{n} \rho_i^l$.

Step 2: Computing δ. As shown in Equation (2), the computation of δ requires to know all points' density values $\Omega = \{\rho_i | \forall i \in \mathbb{P}\}$ as well as the pairwise distance matrix. Similar to the blocking method for computing ρ values, the `map()` function dispatchs the block pairs, and the `reduce()` function computes the pairwise distance values of two input blocks (S_k, S_l) and outputs two sets $\Delta_k^l = \{\delta_i^l | \forall i \in \mathbb{P}_k\}$ where $\delta_i^l = \min_{j | \forall j \in \mathbb{P}_l, \rho_j > \rho_i}(d_{ij})$ and $\Delta_l^k = \{\delta_i^k | \forall i \in \mathbb{P}_l\}$ where $\delta_i^k = \min_{j | \forall j \in \mathbb{P}_k, \rho_j > \rho_i}(d_{ij})$. Besides, each point i's upslope point $u_i^l = \arg\min_{u | \forall u \in \mathbb{P}_l, \rho_u > \rho_i}(d_{iu})$ is also recorded along with its δ_i^l value. Obviously, δ_i^l is not the final result. Another

TABLE 1
Notations

Notation	Definition
S	the set of points
$\mathbb{P}$	the set of point ids
Ω	the set of ρ values, $\Omega = \{\rho_i \| \forall i \in \mathbb{P}\}$
Δ	the set of δ values, $\Delta = \{\delta_i \| \forall i \in \mathbb{P}\}$
$\hat{\Omega}$	the set of approx. ρ values, $\hat{\Omega} = \{\hat{\rho}_i \| \forall i \in \mathbb{P}\}$
$\hat{\Delta}$	the set of approx. δ values, $\hat{\Delta} = \{\hat{\delta}_i \| \forall i \in \mathbb{P}\}$
$\mathcal{P}(S)$	an LSH partition of S
n	the number of subsets
i or j	the point id
k or l	the subset index
M	the number of LSH partition layouts
m	the index of LSH partition layout
c	the cluster id
u_i	point i's upslope point id

1. Please refer to the implementation code of [1].

Authorized licensed use limited to: Northeastern University. Downloaded on April 06,2025 at 03:23:19 UTC from IEEE Xplore. Restrictions apply.

MapReduce job is required to select the smallest δ_i value among the candidates δ_i^l, i.e., $\delta_i = \min_l \delta_i^l$, and to record its corresponding upslope point.

Step 3: Density Peaks Selection and Point Assignment. As shown in Fig. 1c, the decision graph plays a key role on density peaks selection. A point i is depicted on the decision graph by using (ρ_i, δ_i) as its x-y coordinate. The density peaks are identified as outliers in the top right of decision graph. However, drawing a visible figure with millions of points is not feasible. To address this problem, we combine a set of close points with small (ρ_i, δ_i) as a supernode but leaving the points with large (ρ_i, δ_i) drawn separately since only the points with large (ρ_i, δ_i) could be considered as density peaks. Note that it is possible to design certain criteria for choosing the peaks automatically. However, we believe it is better to retain this user-algorithm interaction, since the visualized reference (i.e., decision graph) provides users with an opportunity to better understand the data and choose the preferred clustering result. This is a key feature that distinguishes DP from other clustering algorithms (e.g., Kmeans and DBSCAN), which require users to face the challenge of specifying key algorithm parameters in advance.

Given the chosen density peaks (i.e., cluster centers), we follow the upslope point for each point to assign it to a cluster as illustrated in Fig. 1d. Each point is embedded with five pieces of information including point id i, ρ_i, δ_i, upslope point id u_i, assigned cluster id c, i.e., $\langle i, \rho_i, \delta_i, u_i, c\rangle$, and these points are stored using a fixed-size array structure. To achieve efficient implementation, the points are assigned to a cluster in the descending order of their ρ values. The density peaks with highest ρ values are first labeled with cluster ids. Since a point's upslope point must have a larger ρ and its upslope point should already be assigned to a cluster, the point is simply assigned to the cluster where its upslope point belongs. By this way, the point assignment process is achieved by a single pass of these points. Commonly, this step can be done in memory in a centralized manner.[2] As the data size exceeds the memory space limit, it is easy to implement a disk-based or a distributed version since only one pass of the data is required.

3.2 Cost Analysis and Improvement Opportunities

From the above description, we see that the most expensive steps in Basic-DDP are the computation of ρ (Step 1) and δ (Step 2). The blocking technique still has to send every point $\lceil \frac{n+1}{2} \rceil$ times during the shuffling phase in Step 1 as well as in Step 2. This incurs significant shuffle overhead especially when the point set S is large. Moreover, Basic-DDP computes $\frac{|S|(|S|-1)}{2}$ distances in both Step 1 and Step 2. The computational cost is quadratic with respect to the total number of points $|S|$.

To improve performance, an ideal strategy is to partition S into n disjoint subsets $\{S_k | 1 \leq k \leq n\}$ such that the ρ and δ computation could be self-contained within each partition. First, distances are computed only inside a partition, $\{d_{ij} | \forall i \in \mathbb{P}_k, \forall j \in \mathbb{P}_k\}$. Second, to guarantee the correctness of ρ, the subset S_k must contain each point i's d_c-length neighbors. We have to put more points into every S_k to form S'_k, i.e., $\mathbb{P}'_k = \mathbb{P}_k \cup \{j | \forall i \in \mathbb{P}_k, d_{ij} < d_c\}$. Note that, $S'_k \cup S'_l$ is typically not empty. Recent technology in kNN search [14] or triangle inequality might be employed to select additional points to include in each S'_k. However, the δ computation becomes infeasible. Each point i in S_k is only aware of the distance to the subset S'_k of points, i.e., $\{d_{i,j} | \forall j \in \mathbb{P}'_k\}$. The points with higher density are likely not in S'_k. Copying all the higher density points will incur excessive shuffle cost.

While the above ideal partitioning approach does not work, it inspires us to develop an alternative approximate solution, as will be detailed in the next section.

4 LSH Based Approach

In this section, we propose an approximate distributed algorithm, LSH-DDP, for DP. Intuitively, a locality preserving partition strategy is desirable for DP. This is because the computation of ρ_i is based on the neighbors within a distance of d_c from point i, and the computation of δ looks for the nearest point with higher density. Hence, closer points play a more important role in the computation. As suggested by the name, LSH-DDP leverages Locality-Sensitive Hashing [15] to partition points so that closer points are more likely to be assigned to the same partitions.

To improve approximation accuracy, we partition the point set S using M LSH partition layouts, $\mathcal{P}_1(S), \mathcal{P}_2(S), \ldots, \mathcal{P}_M(S)$. An LSH partition layout $\mathcal{P}_m(S)$ is a partition of the data space. S is split into multiple partitions such that $\mathcal{P}_m(S) = S_1^m \cup S_2^m \cup \ldots$, where $S_k^m \cap S_l^m = \emptyset (\forall k \neq l)$. With a larger M, it is more likely that points that are close will collide in the same partition in at least one partition layouts.

LSH-DDP computes the distances of pairs of points within each partition S_k^m, and derives a set $\hat{\Omega}_k^m$ of approximate ρ values within partition S_k^m. The computation on multiple $\mathcal{P}_m(S)$ can be performed in parallel. Then, LSH-DDP aggregates the multiple approximations, $\hat{\Omega}_k^m$, to obtain more accurate results, $\hat{\Omega}$. The approximation of δ values follows the same strategy. With the previously approximated $\hat{\rho}_i$ values, LSH-DDP finds the upslope point u_i for each point i and computes $\hat{\delta}_i$ within each partition. The multiple approximations, $\hat{\Delta}_k^m$, are further aggregated to obtain more accurate results, $\hat{\Delta}$.

4.1 Step 0: LSH Partition

LSH Background. The Locality-Sensitive Hashing function has the property that points that are closer to each other have a higher probability of colliding than points that are farther apart [15]. It has been widely adopted in solving approximate nearest neighbor search problem [16], [17], [18], [19], [20].

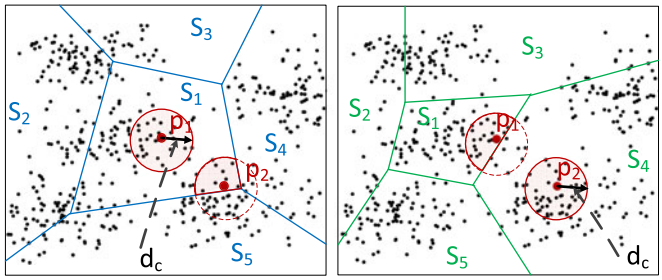
The commonly used LSH function for euclidean distance is as follows [12]

$$h(p) = \left\lfloor \frac{a \cdot p + b}{w} \right\rfloor, \tag{3}$$

where a is a d-dimensional random vector, each entry of which is chosen independently from a p-stable distribution

2. A mid-range server with 32GB memory can process up to 1.6 billion points.

Authorized licensed use limited to: Northeastern University. Downloaded on April 06,2025 at 03:23:19 UTC from IEEE Xplore. Restrictions apply.



(a) LSH partition layout 1 (b) LSH partition layout 2

Fig. 2. ρ computation in two partition layouts (in plane view).

[21], b is a real number chosen from $[0, w]$, and w is also a real number representing the width of the LSH function.

The distance-preserving property of LSH allows us to partition the set of points based on their hash values. If two points i and j are hashed to the same bucket, we know that i and j are close to each other with certain confidence. Therefore, we can assign them to the same partition. However, it is possible that two distant points happen to be hashed to the same bucket according to Equation (3). To reduce such *false positives*, a group G of π hash functions $G = (h_1, h_2, \ldots, h_\pi)$ are employed. That is, only points sharing all the π hash values are placed in the same partition. Thus, each point i is labeled with $G(p_i) = [h_1(p_i), h_2(p_i), \ldots, h_\pi(p_i)]$, which is considered as a partition id. Multiple partitions are formed and assigned to multiple workers[3] for parallel processing. The resulting data partition result is referred to as an *LSH partition layout* $\mathcal{P}$. The formal definition of LSH partition layout is as follows:

Definition 1 (LSH Partition Layout). *Given a set of points S, and a group of hash functions $G = (h_1, h_2, \ldots, h_\pi)$, an LSH partition layout is obtained by hashing every point $p_i \in S$ using G and assigning p_i to the partition as identified by hash key $G(p_i) = [h_1(p_i), h_2(p_i), \ldots, h_\pi(p_i)]$. The point set S is accordingly split into multiple disjoint subsets, i.e., $\mathcal{P}(S) = S_1 \cup S_2 \cup \ldots$, where $S_k \cap S_l = \emptyset, \forall k \neq l$.*

However, it is also possible that points that are close happen to be hashed to different partitions, especially when π is large, incurring *false negatives*. To reduce the number of false negatives, we employ a combination of M hash groups, $(G_1, G_2, \ldots, G_M)$. That is, the point set is partitioned in M different ways. Suppose by applying a hash group G_m, we obtain an LSH partition $\mathcal{P}_m(S) = S_1^m \cup S_2^m \cup \ldots$, where $S_k^m \cap S_l^m = \emptyset, \forall k \neq l$. Similarly, by applying M groups of hash functions $(G_1, G_2, \ldots, G_M)$, we will have M LSH partitions $(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_M)$ of the set S. For example, Fig. 2 illustrates two possible LSH partitions of point set S.[4]

We achieve multiple LSH partition layouts in the map phase. The `map()` function invocation on a point p_i computes M hash keys $G_1(p_i), G_2(p_i), \ldots, G_M(p_i)$ and then sends the intermediate key-value pairs, $\langle G_1(p_i), p_i\rangle, \langle G_2(p_i), p_i\rangle, \ldots, \langle G_M(p_i), p_i\rangle$, to reducers. Each `reduce()` function will receive a subset S_k^m of points under a certain LSH partition layout $\mathcal{P}_m(S)$. In this way, M LSH partition layouts are created. We can also estimate d_c through sampling in the same MapReduce job to save cost.

3. That is, each hash bucket is assigned to a distributed computation unit, e.g., a reducer. Here, we use the term worker for ease of exposition.

4. Note that, Figs. 2 and 4 are only illustrative figures. The real LSH partition might not be linearly separable. The number of partitions is dependent on the LSH parameter w and π as will be discussed in Section. 5.2

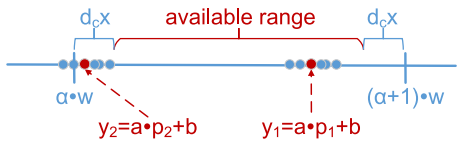


Fig. 3. Graphic interpretation of Lemma 1.

4.2 Step 1: Approximating ρ

Local Computation of $\hat{\rho}_i^m$. For a certain LSH partition $\mathcal{P}_m(S)$, a subset S_k^m is shuffled to a `reduce()` function. The `reduce()` function first computes the distances between any pairs of points in S_k^m. Then it computes a density value $\hat{\rho}_i^m$ for each point i, i.e., $\hat{\rho}_i^m = \sum_{j|j\in\mathbb{P}_k^m} \chi(d_{ij} - d_c)$.

However, $\hat{\rho}_i^m$ is not necessarily equal to ρ_i. As shown in Fig. 2a, point p_2 in LSH partition layout 1 is located near the border line between S_1, S_4, and S_5. The computation of $\hat{\rho}_2^1$ is limited only to the points that are in S_1. However, it is clear that a large number of points that are close to p_2 are located in S_4 and S_5. Therefore, $\hat{\rho}_2^1 < \rho$. The use of multiple hash groups mitigates the problem. As shown in Fig. 2b, all p_2's d_c-length neighbors reside in the same partition as p_2. Therefore, $\hat{\rho}_2^2 = \rho$.

To study the probability of $\Pr[\hat{\rho}_i^m = \rho_i]$, we give the following two lemmas.

Lemma 1. *Given a point p_i and an LSH function $h(p_i) = \lfloor \frac{a\cdot p_i + b}{w} \rfloor$, for the points $\{p_j | j \in \mathbb{P}, d_{ij} \leq d_c\}$, the probability that all these points are hashed to the same bucket is as follows:*

$$P_\rho(w, d_c) = \Pr\big[h(p_i) = h(p_j), \forall j \in \mathbb{P}, d_{ij} \leq d_c\big] \geq 1 - \frac{4d_c}{\sqrt{2\pi}w}. \quad (4)$$

Proof. Fig. 3 depicts the idea of the proof intuitively. Let us consider a number line, where each point is a real number. $y_i = a \cdot p_i + b$ is a point on the number line. By floor dividing w, the number line is divided into a sequence of w-width slots. According to the LSH function, all the points in the same w-width slot share the the same hash key. The points that are close to p_i are all hashed to the points close to y_i on the number line. The position of y_i is important. If y_i is close to the center of the slot, it is more likely that all d_c-length neighbors of p_i are in the same slot.

According to the definition of p-stable distribution [12], given a d-dimensional random vector a each entry of which is chosen independently from a standard gaussian distribution $\mathcal{N}(0, 1)$, for two points p_i and p_j, the distance between their projections $|a \cdot p_i - a \cdot p_j|$ is distributed as $d_{ij}x$, where x is the *absolute value* of a standard gaussian random variable. Therefore, for any p_j where $d_{ij} < d_c$, we have $\max_j |y_i - y_j| = \max_j |a \cdot p_i - a \cdot p_j| < d_c x$.

Moreover, $y_i = a \cdot p_i + b$ is uniformly distributed in a certain slot. To ensure that y_i and all its d_c-length neighbors are in the same slot, y_i has to be located in the interval of $[\alpha w + d_c x, (\alpha + 1)w - d_c x)$ for some α, as shown in

Authorized licensed use limited to: Northeastern University. Downloaded on April 06,2025 at 03:23:19 UTC from IEEE Xplore. Restrictions apply.

Fig. 3. The probability that y_i resides in such an interval is $\frac{w-2d_cx}{w} = 1 - \frac{2d_cx}{w}$. The probability density function of the absolute value of the standard gaussian distribution is $f_p(x) = \frac{2e^{-x^2/2}}{\sqrt{2\pi}}$, where $x \geq 0$. Therefore, the probability becomes $1 - \frac{2d_cx}{w} = \int_0^{\infty}(1 - \frac{2d_cx}{w})f_p(x)dx$, and a further calculation shows that the probability is $1 - \frac{4d_c}{\sqrt{2\pi}w}$. □

Further, it is obvious to obtain Lemma 2. The proof can be found in Section 1 of the supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2016.2609423.

Lemma 2. *For an LSH partition $\mathcal{P}_m$ with π hash functions, we have* $\Pr[\hat{\rho}_i^m = \rho_i] \geq P_\rho(w, d_c)^\pi$.

Aggregation of Multiple $\hat{\rho}_i^m$. The point set is partitioned in M LSH partition layouts $(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_M)$. Accordingly, for each point i, we will obtain M approximate density values $(\hat{\rho}_i^1, \hat{\rho}_i^2, \ldots, \hat{\rho}_i^M)$. These candidates (that are retrieved from multiple distributed reducers) are aggregated in the second MapReduce job. Since $\hat{\rho}_i^m \leq \rho$, we choose $\hat{\rho}_i = \max_m \hat{\rho}_i^m$. We hope that the aggregate value is closer to the exact value.

Employing M LSH partitions reduces the chances that a point's d_c-length neighbors reside in different partitions. As a result, it reduces the number of false negatives, and thus significantly increases $\Pr[\hat{\rho}_i = \rho_i]$.

Theorem 1. *With M LSH partitions $(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_M)$, we have* $\Pr[\hat{\rho}_i = \rho_i] \geq 1 - \left[1 - P_\rho(w, d_c)^\pi\right]^M$.

Proof. $\hat{\rho}_i^m \leq \max_m \hat{\rho}_i^m \leq \hat{\rho}_i$. If $\max_m \hat{\rho}_i^m \neq \rho_i$, then $\forall m = 1, \ldots, M$, $\hat{\rho}_i^m \neq \rho_i$. From Lemma 2, $\Pr[\hat{\rho}_i^m = \rho_i] \geq P_\rho(w, d_c)^\pi$. Since $G_m (1 \leq m \leq M)$ is independently and randomly generated, we have the following:

$$\begin{aligned}\Pr[\hat{\rho}_i = \rho_i] &= 1 - \prod_{m=1}^{M}(1 - \Pr[\rho_i^m = \rho_i]) \\ &\geq 1 - \left[1 - P_\rho(w, d_c)^\pi\right]^M.\end{aligned}$$

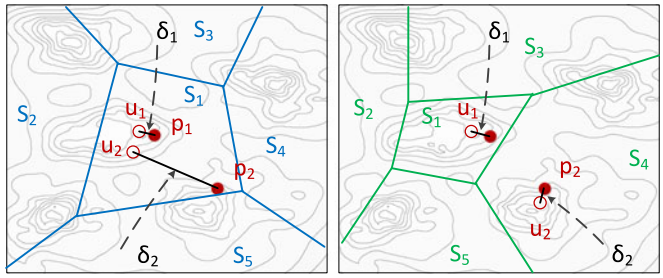
□

4.3 Step 2: Approximating δ

The computation of δ depends on ρ values. Therefore, after Step 2, LSH-DDP associates each point p_i with its approximate $\hat{\rho}_i$ value. Then, Step 3 follows the same idea as approximating ρ. LSH-DDP partitions the points using M LSH layouts $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_M$ with the `map()` function. Then it performs local computation for δ_i^m values as follows.

Local Computation of δ_i^m. Let us consider a `reduce()` function working on a partition S_k^m in a certain LSH partition layout $\mathcal{P}_m$. LSH-DDP computes the distances between all pairs of points in S_k^m. Then, using the approximate density $\{\hat{\rho}_j | j \in \mathbb{P}_k^m\}$, it approximates $\hat{\delta}_i^m = \min_{j|j \in \mathbb{P}_k^m, \hat{\rho}_j > \hat{\rho}_i}(d_{ij})$ for any $i \in \mathbb{P}_k^m$. For the point with the highest density in S_k^m, i.e., point $i = \arg\max_{i|i \in \mathbb{P}_k^m} \hat{\rho}_i$, we set $\hat{\delta}_i^m = \infty$.

However, even though the approximated $\hat{\rho}_i$ were exactly equal to ρ_i, δ_i^m might not be equal to δ_i, since the computation is constrained within a subset of points. For example, in LSH partition layout 1 as shown in Fig. 4a, since point p_2's real upslope point resides in a different partition, the local δ approximation returns a wrong result, an incorrect upslope point on another density mountain. Fortunately, in LSH partition layout 2 as shown in Fig. 4b, p_2 and its upslope point are assigned in the same partition, and the correct δ_2 can be computed.



(a) LSH partition layout 1 (b) LSH partition layout 2

Fig. 4. δ computation in two partition layouts (in contour view).

Assume $\hat{\rho}_i = \rho_i$, we study the probability of $\Pr[\hat{\delta}_i^m = \delta_i]$ in the following lemmas. First, based on the property of LSH and p-stable distribution (refer to Datar's paper [12]), we have Lemma 3. Further, we have Lemma 4 based on LSH properties, and the proof can be found in Section 2 of the supplementary file, available online.

Lemma 3. *Given a point p_i and an LSH function $h(p_i) = \lfloor \frac{a \cdot p_i + b}{w} \rfloor$, suppose p_i's upslope point is p_{u_i} (if exist) and d_{iu_i} is the distance from i to u_i (i.e., $d_{iu_i} = \delta_i$), we have*

$$\begin{aligned}P_{\delta_i}(d_{iu_i}, w) &= \Pr\left[h(p_i) = h(p_{u_i})\right] \\ &= \int_0^w \frac{1}{d_{iu_i}} f_p\left(\frac{x}{d_{iu_i}}\right)\left(1 - \frac{x}{w}\right)dx \\ &= 2norm\left(\frac{w}{d_{iu_i}}\right) - 1 - \frac{2d_{iu_i}}{\sqrt{2\pi}w}\left(1 - e^{-\frac{w^2}{2d_{iu_i}^2}}\right),\end{aligned}$$

where $f_p(x)$ denotes the probability density function of the absolute value of a standard gaussian distribution, and $norm(\cdot)$ is the cumulative distribution function (cdf) for a random variable that is distributed as $\mathcal{N}(0, 1)$.

Lemma 4. *Suppose point p_i's real upslope point is p_{u_i} (if exist), by a certain LSH partition $\mathcal{P}_m$ with π hash functions, we have* $\Pr[\hat{\delta}_i^m = \delta_i] = P_{\delta_i}(d_{iu_i}, w)^\pi$

Aggregation of Multiple $\hat{\delta}_i^m$. For each point i, we will obtain M approximate values $(\hat{\delta}_i^1, \hat{\delta}_i^2, \ldots, \hat{\delta}_i^M)$ in various LSH layouts. We hope that at least one of them is equal or close to the exact value. According to Equation (2), the smallest one is more likely to be the exact δ_i. Therefore, we aggregate these approximate $(\hat{\delta}_i^1, \hat{\delta}_i^2, \ldots, \hat{\delta}_i^M)$ in a MapReduce job and set $\hat{\delta}_i = \min_m \hat{\delta}_i^m$. Similar to ρ approximation, the probability $\Pr[\hat{\delta}_i = \delta_i]$ is enlarged as follows.

Theorem 2. *Given a point i's upslope point u_i, with M LSH partitions $(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_M)$, we have* $\Pr[\hat{\delta}_i = \delta_i] = 1 - \left[1 - P_{\delta_i}(d_{iu_i}, w)^\pi\right]^M$.

4.4 Step 3: Further Correction of $\hat{\delta}$

From Theorem 2, we can see that the probability $\Pr[\hat{\delta}_i = \delta_i]$ highly depends on d_{iu_i} or δ_i, i.e., the distance from point i to its "nearest" neighbor with higher density. Generally

Authorized licensed use limited to: Northeastern University. Downloaded on April 06,2025 at 03:23:19 UTC from IEEE Xplore. Restrictions apply.

speaking, as shown in Fig. 1d, d_{iu_i} is small for most points and therefore the probability $\Pr[\hat{\delta}_i = \delta_i]$ is correspondingly high for most points. However, this is not true if point i is distant from its upslope point (i.e., d_{iu_i} or δ_i is large). This leads to a very interesting situation. $\hat{\delta}_i$ is more accurate for smaller δ_i but inaccurate for larger δ_i.

Furthermore, since the density peaks are with large δ_i, they are distant from each other and are unlikely to be hashed to the same bucket under a locality-preserving hash function. Therefore, LSH-DDP may wrongly recognize these density peaks as the absolute density peak in a partition and therefore assign $\hat{\delta}_i = \infty$. Although these points are very likely to be the local density peaks and also probably be chosen as density peaks in the density peak selection step, a few wrong selections of density peaks will change the cluster result and result in more fine-grained clusters.

To rectify these $\hat{\delta}_i$ values, we should first find the points whose $\hat{\delta}_i$ are highly likely to be wrongly approximated, i.e., $\{p_i|\hat{\delta}_i \neq \delta_i\}$. Suppose ρ is correctly approximated. Given a lower bound accuracy $\mathcal{A}_\delta$ of the δ approximation, we have

$$\Pr[\hat{\delta}_i = \delta_i] = 1 - \left[1 - P_{\delta_i}(d_{iu_i}, w)^\pi\right]^M \geq \mathcal{A}_\delta. \tag{5}$$

Considering the representation of $P_{\delta_i}(d_{iu_i}, w)$ in Lemma 3, d_{iu_i} is the only variable in (5). By solving this equation (using Trust-Region with DogLeg method [22]), we can obtain the minimum d_{iu_i} that satisfies this lower bound accuracy requirement , i.e., $d_{iu_i} \geq \gamma$ where γ is solution of the equation and the lower bound distance to its upslope point. Furthermore, point i's δ_i is the distance to its nearest neighbor with higher density. Its approximation $\hat{\delta}_i$ cannot be less than its real value δ_i, i.e., $\hat{\delta}_i \geq \delta_i$ or $d_{i\hat{u}_i} \geq d_{iu_i}$ (where $\hat{u}_i$ is observed upslope point by approximation). We have $d_{i\hat{u}_i} \geq d_{iu_i} \geq \gamma$. With the probability guarantee, if the observed $d_{i\hat{u}_i}$ is larger than γ, the approximation of δ_i are more likely inaccurate and should be further rectified. Therefore, our further correction step will rectify these $\hat{\delta}_i$ values whose $\hat{\delta}_i = d_{i\hat{u}_i} \geq \gamma$.

However, precisely rectifying these $\hat{\delta}_i$ requires to compute point i's distance to all the points with higher density. This results in significant computational and communication cost (if implemented distributively). Instead, we rely on rough rectification. Given that δ_i computation only considers the distance to higher density points, we sample the points with larger $\hat{\rho}$ values. The larger a point's $\hat{\rho}_i$ value is, the higher probability the point is sampled. Only these sampled points are considered as the candidate points for distance measurement when rectifying $\hat{\delta}_i$ values. Though this approach is simple, our empirical results show that it is effective enough.

4.5 MapReduce Implementation

To sum up, the MapReduce implementation of LSH-DDP consists of five MapReduce jobs and a centralized program (for density peak selection and point assignment). The first job performs LSH partition (Map1) and local computation of $\hat{\rho}_i^m$ (Reduce1). The second job aggregates the $\hat{\rho}_i^m$ values (Reduce2). Similarly, LSH partition (Map3) and local computation of $\hat{\delta}_i^m$ (Reduce3) are carried out in the third job. The fourth job aggregates the $\hat{\delta}_i^m$ values (Reduce4). The centralized program first drafts a decision graph based on the obtained $\hat{\rho}$ and $\hat{\delta}$ values and then let users select a set of density peaks with $\hat{\rho}$ value at least ρ_{peak} and $\hat{\delta}$ value at least δ_{peak}, i.e., $\{p_i|\hat{\rho}_i > \rho_{peak}, \hat{\delta}_i > \delta_{peak}\}$. The fifth job further corrects $\hat{\delta}$ values. The points whose $\hat{\delta}_i \geq \gamma$ are filtered for further correction (Map5). At the same time, the points are sampled with probability $\frac{1}{1+e^{(\hat{\rho}_i - \rho_{peak})}}$ if $\hat{\rho}_i \geq \rho_{peak}$ and $\beta \cdot \frac{1}{1+e^{(\hat{\rho}_i - \rho_{peak})}}$ if $\hat{\rho}_i < \rho_{peak}$ where β is a given sample rate (Map5). For each to-be-corrected point i, its distance to higher density sampled points are measured, and the $\hat{\delta}_i$ is updated once a smaller $\hat{\delta}_i$ is found (Reduce5). Finally, the centralized density peak selection and point assignment step is re-performed with the corrected $\hat{\delta}$ values.

5 Parameters Tuning

To launch LSH-DDP, there are three parameters to be determined, the number of hash groups M, the number of hash functions in each group π, and the width of hash function w. A reasonable selection of these parameters is crucial to approximation accuracy and performance. The determination of these three parameters is an optimization problem, which takes two factors into account: the accuracy of result and the cost (including shuffle cost and computational cost). In this section, we discuss the parameter determination with a certain accuracy expectation.

5.1 Problem Formulation

Accuracy. The LSH-DDP algorithm makes the cluster assignment for each point based on their approximated $\hat{\rho}$ and $\hat{\delta}$ values. The accuracy of each point assignment should be $\Pr[\hat{\rho}_i = \rho_i] \cdot \Pr[\hat{\delta}_i = \delta_i] = \mathcal{A}_\rho \cdot \mathcal{A}_\delta$. However, according to Theorem 2 the accuracy of an approximated $\hat{\delta}_i$ greatly depends on the real δ_i, which is unknown in advance. We would like to only analyze the accuracy of the approximated $\hat{\rho}$ values. According to Theorem 1, we define the *expected accuracy* as follows:

$$\mathcal{A}_\rho(w, \pi, M) = 1 - \left[1 - P_\rho(w, d_c)^\pi\right]^M, \tag{6}$$

where $P_\rho(w, d_c)$ is defined in Equation (4) and d_c is fixed.

Shuffle Cost. For M LSH partition layouts, LSH-DDP shuffles M copies of each point in the ρ approximation and δ approximation, respectively, which is $2M \cdot |S|$. It also aggregates $\hat{\Omega}^m, m = 1, 2, \ldots, M$ and $\hat{\Delta}^m, m = 1, 2, \ldots, M$. But since in general either $|\hat{\Omega}|$ or $|\hat{\Delta}|$ is much smaller than $|S|$, the shuffle cost of $\hat{\rho}$ values set $\hat{\Delta}$ and $\hat{\delta}$ values set $\hat{\Omega}$ can be ignored. In the correction step, since the $\hat{\delta}_i$ values of only a subset of points should be corrected and only a small portion of high density points are sampled, the shuffle cost of the rectification job are ignored for simplicity. Therefore, the *shuffle cost.* (or the size of shuffled data) can be simplified as

$$\mathcal{C}_s(w, \pi, M) = 2M \cdot |S|. \tag{7}$$

Computational Cost. The computations occur in the LSH partition, the distance calculations, and $\hat{\rho}$ and $\hat{\delta}$ approximations. Among them, the distance calculations are the most

Authorized licensed use limited to: Northeastern University. Downloaded on April 06,2025 at 03:23:19 UTC from IEEE Xplore. Restrictions apply.

costly. We consider the number of distance calculations in the ρ and δ approximation steps as the computational cost, which are from the generations of distributed distance matrices $D_{kk}^m, \forall k, 1 \le m \le M$ where D_{kk}^m is the pair-wise distance matrix of subset S_k^m and is a $N_k^m \times N_k^m$ matrix (N_k^m is the number of points in S_k^m). Therefore, the expected *computational cost* can be represented as

$$E\big[\mathcal{C}_c(w,\pi,M)\big] = E\left[\sum_{m=1}^{M}\sum_{k=1}^{K}\left(N_k^m\right)^2\right] = M\cdot\sum_{k=1}^{K}{N_k}^2, \quad (8)$$

where K is the expected number of partitions and $\sum_{k=1}^{K} N_k = N$.

We aim to minimize both the shuffle cost and the computational cost while satisfying a certain accuracy guarantee. This is a multi-objective optimization problem. Apparently, this multi-objective optimization problem can be transformed to a single objective optimization problem by unifying the costs into time cost. Suppose the ratio of the time unit for shuffling each byte to the time unit for each distance calculation is μ, which varies for different clusters and can be estimated in a MapReduce job (e.g., [0.02-0.1] for our local cluster and [0.1-0.3] for our EC2 cluster). The single optimization problem can be described as follows:

$$\begin{aligned} \min.\quad & \mu\cdot 2M\cdot|S| + M\cdot\sum_{k=1}^{K}{N_k}^2 \\ \text{s.t.}\quad & 1-\left[1-P_\rho(w,d_c)^\pi\right]^M \ge \text{required value}. \end{aligned} \quad (9)$$

We can see that the parameters w, π, M play a key role in solving this optimization problem.

5.2 Analysis of Parameter Variations

From Theorem 1, it is obvious that the accuracy increases with the increase of M and w, and with the decrease of π.

For both shuffle and computational cost, it is obvious that they increase with the increase of M. The computational cost also increases with the increase of the sum of squares $\sum_{k=1}^{K}{N_k}^2$, where $\sum_{k=1}^{K} N_k = N$. The value of $\sum_{k=1}^{K}{N_k}^2$ depends on the data distribution and affected by w and π. We do not make any assumption of data distribution and would like to study the relationship between $\sum_{k=1}^{K}{N_k}^2$ and the parameters w and π. Intuitively, small w leads to narrow partition, and large π leads to a fine partition of the space. That is, small w and large π lead to a large number of small N_k and probably small $\sum_{k=1}^{K}{N_k}^2$. Therefore, the computational cost should decrease with the decrease of M and w, and with the increase of π.

Based on the above analysis, we see that the impacts of the three parameters on the expected accuracy and on performance are reverse. As a result, there is a tradeoff between approximation accuracy and performance.

5.3 Offline Parameter Tuning

We tune M, π, and w to minimize the runtime while guaranteeing a given accuracy requirement. Since the computational cost (mainly determined by $\sum_k {N_k}^2$) depends on not only LSH parameters $\{M,\pi,w\}$ but also the data distribution, the optimization problem cannot be solved without knowledge of the data. We employ offline parameter tuning by sampling the data points. ω groups of N' points[5] are sampled through distributed reservoir sampling. We perform LSH partitions on these ω samples sets according to various parameter combinations and obtain ω LSH layouts, i.e., ω results of $\sum_k {N'_k}^2$ values.[6] We then compute the average and scale it by $(\frac{N}{N'})^2$ to predict $\sum_k {N_k}^2$, i.e., $\frac{\sum^{\omega}\sum_k {N'_k}^2}{\omega}\cdot(\frac{N}{N'})^2$. We use the following greedy heuristic to look for the optimal parameters set.

First, assume M and π are fixed as M_0 and π_0 respectively. We compute the minimum value of w (i.e., w_0) that satisfies a given accuracy requirement $\mathcal{A}$, i.e., solving equation $1-\left[1-P_\rho(w,d_c)^\pi\right]_0^M = \mathcal{A}$ for the variable w. Therefore, given M_0, π_0 and the predicted $\sum_k {N_k}^2$, the total cost T_0 can be obtained according to Equation (9).

Next, we try $M_{+x} = M_0 + x$ (x is the stepsize and $x \in \mathbb{Z}^+$) as well as $M_{-x} = M_0 - x$ while fixing π_0 and repeat the above process to obtain T_{+x} and T_{-x}. We fix the M that results in smaller total cost, and then try π_{+y} and π_{-y}. Similarly the π with smaller total cost is chosen. We repeat this process by alternatively varying M and π until the total cost T is not decreased no matter increasing or decreasing M and π. The resulted $\{M,\pi,w\}$ are returned as the parameters set.

It is noticeable that the complexities in distributed environment (e.g., load unbalance, synchronization barrier, network congestion, node failure, in-memory or external sort) impact the prediction accuracy. But it can provide a relatively reasonable parameter setting with regard to the data distribution. Our experimental results (Section 6.5) shows its effectiveness. Our empirical study also shows that the runtime is stable when M is large enough, and a recommended parameters setting range is provided.

6 Experimental Evaluation

6.1 Experimental Setup

Machine Configuration. The experiments are performed both on our local cluster of machines and on EC2 cloud. Our local cluster contains 1 master and 4 slave workers, each equipped with an Intel I5-4690 3.3G 4-core CPU, 4 GB memory, running Hadoop 1.2.1. The EC2 cluster consists of 64 m1.medium instances.

Data Sets. Table 2 lists the data sets that we use in our experiments. There are two small sized 2D data sets, four real world medium sized high-dimensional data sets, and three large high-dimensional data sets.[7,8,9] We use the 2D data sets to visualize the clustering results, the medium data sets to evaluate the efficiency in our local cluster, and the large data sets to evaluate the efficiency in large EC2 cluster.

6.2 DP versus Previous Algorithms

Before evaluating LSH-DDP, we would like to understand DP's advantages over previous clustering algoritms. Fig. 5a depicts the ground truth. Figs. 5b, 5c, 5d, 5e, and 5f compare

5. The sample rate can be chosen based on data size and cluster performance.

6. The average of multiple $\sum_k {N'_k}^2$ from multiple random LSH partitions would be more accurate.

7. http://cs.joensuu.fi/sipu/datasets/

8. https://archive.ics.uci.edu/ml/datasets.html

9. http://www.cs.unipaderborn.de/en/fachgebiete/agbloemer/research/clustering/streamkmpp

Authorized licensed use limited to: Northeastern University. Downloaded on April 06,2025 at 03:23:19 UTC from IEEE Xplore. Restrictions apply.

TABLE 2
Data Sets

data set	# instances	# dimensions
Aggregation	788	2
S2	5,000	2
Facial	27,936	300
KDD	145,751	74
3Dspatial	434,874	4
BigCross500K	500,000	57
USCensus	2,458,285	68
BigCross	11,620,300	57
Activity	43,930,257	16

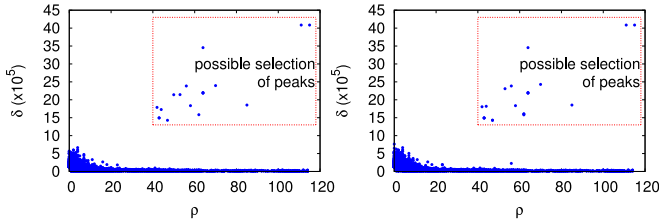
the clustering results of DP and four previous representative algorithms, including agglomerative hierarchical cluster (connectivity-based), K-means (centroid-based), EM (distribution-based), DBSCAN (density-based). Table 3 lists the key features of the clustering algorithms.

The input parameter d_c of DP_Cluster is estimated as the 2 percent position of the ascending ordered distance set (see Section 3.1). For the algorithms that take the number of clusters k as the input parameter, k is set to the number of clusters in the ground truth. DBSCAN's input parameter ε is set to d_c, and the minimum number of points in a cluster is set to 1.

Fig. 5 shows the results for the Aggregation data set, which is a shaped data set. In addition, we compare the algorithms using seven other shaped data sets and see similar trends. For space limitation, we focus only on the Aggregation data set here. There are seven clusters in the ground truth. The hierarchical and the DBSCAN algorithms correctly identify three clusters, but make mistakes for the other clusters. The two algorithms cannot easily separate clusters that are close to each other. On the other hand, K-means and EM can correctly identify four clusters, while they work poorly for non-oval shapes. In contrast, DP correctly identifies all the seven clusters, achieving the best clustering results.

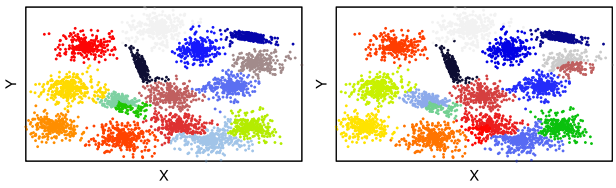
6.3 Effectiveness of LSH-DDP

Visualized Cluster Result. In order to visualize the cluster result, we run Basic-DDP and LSH-DDP on a small sized 2D data set, S2. In LSH-DDP, we set $\mathcal{A} = 0.99, M = 10, \pi = 3$.



(a) Basic-DDP (b) LSH-DDP

Fig. 6. Decision graphs (S2).



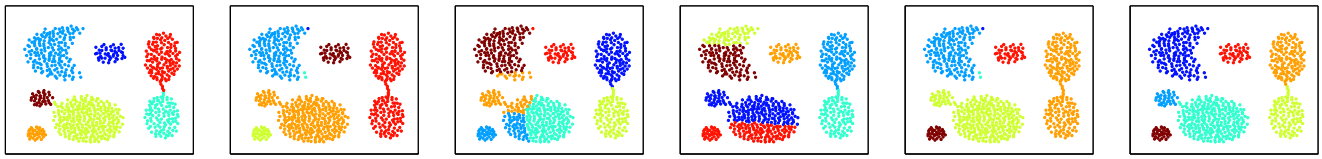
(a) Basic-DDP (b) LSH-DDP

Fig. 7. Cluster result (S2).

Figs. 6a and 6b show the decision graphs for Basic-DDP and LSH-DDP, respectively. The decision graph of Basic-DDP is generated using the computed exact (ρ, δ) values, while the decision graph of LSH-DDP is drawn using the approximate ($\hat{\rho}$, $\hat{\delta}$) values. We show a possible selection of peaks on the two decision graphs (i.e., all points that satisfy $\rho > 40$ and $\delta > 14$).

We see that the decision graphs of Basic-DDP and LSH-DDP are roughly the same. Thanks to the $\hat{\delta}$ correction technique, most of the wrongly approximated $\hat{\delta}_i$ values are corrected. The only difference is that one more peaks is chosen in LSH-DDP decision graph. This is reflected in the cluster result as shown in Fig. 7. One more group of points is clustered in LSH-DDP. However, the cluster results of Basic-DDP and LSH-DDP are almost the same. Differences exist only at boundary points and/or for deciding whether a set of points should be clustered at a finer granularity.

Expected Accuracy of $\hat{\rho}$. We further evaluate the accuracy of LSH-DDP. Since the cluster result of a large multi-dimensional data set cannot be easily visualized, we focus on



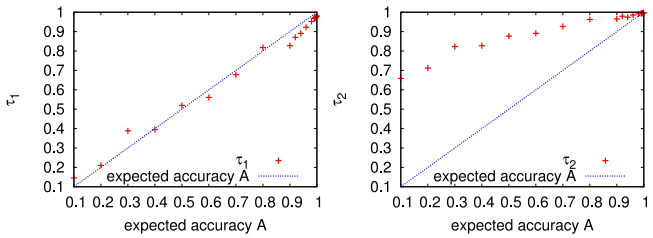
(a) Ground Truth (b) Hierarchical result (c) Kmeans result (d) EM result (e) DBSCAN result (f) DP result

Fig. 5. Cluster results of different algorithms for Aggregation dataset ($k = 7$).

TABLE 3
Key Features of Various Clustering Algorithms

	iterative	cluster shape assumption	predefined # of clusters	complexity	embarrassingly parallel	interactivity
hierarchical	no	yes	no	$O(n^3)$	no	no
k-means	yes	yes	yes	$O(n \cdot k \cdot I)$	yes	no
EM	yes	yes	yes	$O(n \cdot k \cdot I)$	yes	no
DBSCAN	no	no	no	$O(n^2)$	no	no
DP	no	no	no	$O(n^2)$	yes	yes

Authorized licensed use limited to: Northeastern University. Downloaded on April 06,2025 at 03:23:19 UTC from IEEE Xplore. Restrictions apply.



(a) $\mathcal{A}$ vs.τ_1 (b) $\mathcal{A}$ vs.τ_2

Fig. 8. Expected accuracy $\mathcal{A}$ versus τ_1 and τ_2 (S2).

measuring the accuracy of $\hat{\rho}$. We define two metrics τ_1 and τ_2 to characterize the accuracy of the approximation. $\tau_1 = \frac{|\{\hat{\rho}_i | \forall i \in \mathbb{P}, \hat{\rho}_i = \rho_i\}|}{N}$ is the fraction of correctly approximated ρ values. Larger τ_1 means that the $\hat{\rho}$ of more points are approximated correctly. When every $\hat{\rho}$ is approximated correctly, $\tau_1 = 1$. $\tau_2 = 1 - \frac{\sum_i |\rho_i - \hat{\rho}_i|}{\sum_i \rho_i}$. It is 1 minus the normalized absolute error. Hence, the smaller the error, the larger the τ_2. When the error approaches 0, τ_2 grows to 1.

We have run experiments for all the four medium sized data sets and see similar results. For space limitation, we focus on BigCross500K in Fig. 8. On the x-axis, we vary the expected accuracy $\mathcal{A}$. Given a $\mathcal{A}$, we set the LSH-DDP parameters accordingly and then run the algorithm. The resulting τ_1 and τ_2 are reported in Figs. 8a and 8b, respectively. From the figures, we see that both τ_1 and τ_2 increase as the expected accuracy $\mathcal{A}$ increases. Both metrics approach 1 as $\mathcal{A}$ approaches 1. Note that the definition of τ_1 corresponds to the accuracy target. It is clear that τ_1 points reside closely around the diagonal line. This shows that LSH-DDP has successfully realized the accuracy target as specified by $\mathcal{A}$.

Effect of $\hat{\delta}$ Correction. As discussed in Section 4.4, the approximated $\hat{\delta}_i$ is more accurate for smaller δ_i but inaccurate for larger δ_i. Moreover, a few points with large δ values could be misidentified as local density peaks. This might lead to more clusters than expected. In order to rectify the wrongly approximated $\hat{\delta}$ values and help identify the real density peaks, we propose $\hat{\delta}$ correction technique.

To verify the effect of the $\hat{\delta}$ correction step, Table 4 shows the ratio of wrongly approximated $\hat{\delta}$ values and the number of local density peaks (i.e., no other denser points found after LSH partition) before and after correction. We can see that large amount of wrongly approximated $\hat{\delta}$ values are corrected. Furthermore, all the misidentified density peaks are rectified.

TABLE 4
Effect of $\hat{\delta}$ Correction

dataset	before		after	
	wrong $\hat{\delta}$	# peaks	wrong $\hat{\delta}$	# peaks
Facial	14.92%	129	10.72%	1
KDD	6.75%	271	3.36%	1
BigCross500K	4.86%	43	1.45%	1

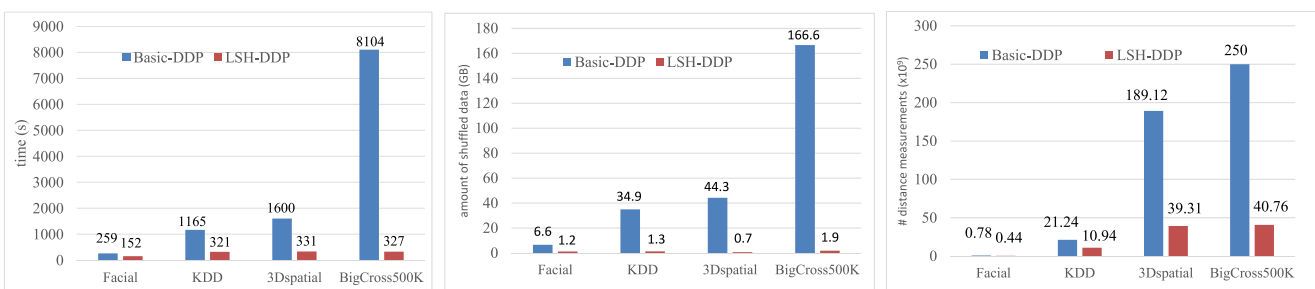
6.4 Efficiency of LSH-DDP

Runtime. We run Basic-DDP and LSH-DDP on four data sets, i.e., Facial, KDD, 3Dspatial, and BigCross500K on the local cluster of machines. The parameters of LSH-DDP are set as follows: $\mathcal{A} = 0.99, M = 10, \pi = 3$, and the block size parameter of Basic-DDP is set as 500. As shown in Fig. 9a, LSH-DDP is dramatically better than Basic-DDP, achieving 1.7–24× speedups. Moreover, the larger the data set size, the more benefit LSH-DDP brings. To understand the performance benefit, we delve into the communication cost and the computation cost in the following.

Shuffle Cost. Fig. 9b compares the total amount of data shuffled in the MapReduce jobs of Basic-DDP and LSH-DDP. Basic-DDP has to send every point to every other point using the Map and the shuffle stages. In contrast, LSH-DDP sends only the local results computed from LSH local partitions, thereby avoiding the quadratic communication cost. As shown in Fig. 9b, LSH-DDP reduces the amount of shuffled data by 5–87× compared to Basic-DDP. Since the amount of shuffled data in Basic-DDP grows quadratically, LSH-DDP sees larger savings for larger data sets.

Computational Cost. Fig. 9c reports the number of distance measurements computed in Basic-DDP versus LSH-DDP. The computation cost of Basic-DDP grows quadratically, while LSH-DDP sees only linear growth. Consequently, the savings of LSH-DDP grow as the input data set size increases. We see a 1.7–6.1× savings for computational cost.

Comparison to EDDPC. EDDPC [23] is a recently work on parallelizing DP algorithm. It leverages Voronoi diagram and careful data replication/filtering to reduce the huge amount of useless distance measurement cost and data shuffle cost. Rather than approximation, EDDPC will return the exact ρ and δ values. We also compare our approach with EDDPC for clustering the BigCross500K data set. The results are listed in Table 5. We can see that LSH-DDP requires less runtime and much less shuffled data though higher number of distance measurements (i.e., # dist.). Note



(a) Runtime (b) Shuffle cost (c) Computational cost

Fig. 9. Basic-DDP versus LSH-DDP for different data sets.

Authorized licensed use limited to: Northeastern University. Downloaded on April 06,2025 at 03:23:19 UTC from IEEE Xplore. Restrictions apply.

TABLE 5
Comparison to EDDPC on BigCross500K

	runtime (s)	shuffle (GB)	# dist. ($\times 10^9$)
Basic-DDP	8,104	166.6	250
EDDPC	667	5.7	13.3
LSH-DDP	327	1.9	40.7

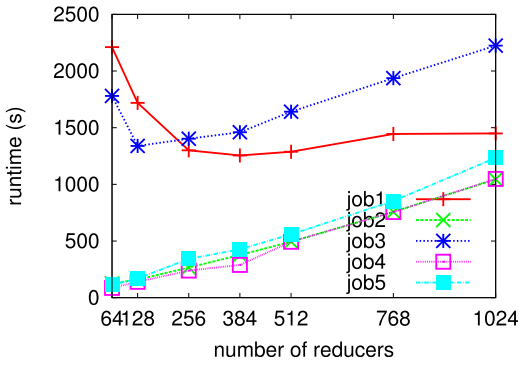


Fig. 10. Runtime of LSH-DDP when varying the number of reducers (BigCross).

that, LSH-DDP will result in even higher efficiency with lower accuracy requirement.

Large Scale Experiments on EC2. In order to see the performance of LSH-DDP for very large data sets in large scale distributed environments, we run the algorithms on the three large data sets (USCensus, BigCross, Activity) on our EC2 cluster.

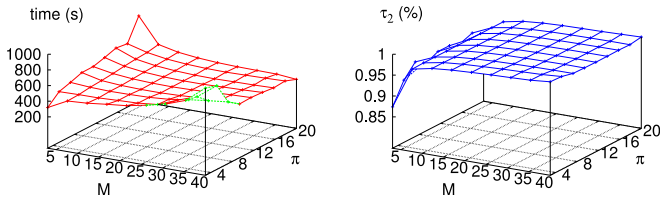
As known, the number of reducers plays a key role in distributed computing performance. Large number of reducers helps increase parallelization but results in more task initialization overhead. Fig. 10 shows the runtime of LSH-DDP on the BigCross data set when varying the number of reducers. Job 1 and Job 3 perform LSH partition and local approximations of $\hat{\rho}$ values and $\hat{\delta}$ values respectively, which are relatively heavy loaded. While Job 2 and Job 4 only aggregate the local approximations of $\hat{\rho}$ values and $\hat{\delta}$ values respectively, which are relatively light loaded. Job 5 further corrects the $\hat{\delta}$ values, which is also light loaded. We can see that it is preferable to choose a larger number of reducers for heavy loaded job while a smaller number of reducers for light loaded job. Once the overhead of task initializations overwhelms the potential benefit of parallelization, the runtime prolongs with the increase of number of reducers.

Table 6[10] shows the runtime of LSH-DDP and Basic-DDP on different data sets. We only consider the runtime of the ρ/δ computation runtime for LSH-DDP and Basic-DDP. The number of reducers is 256 for job1/job3 and 64 for job2/job4/job5. The runtime of 5 jobs is listed separately. The parameters settings are as follows: $\mathcal{A} = 0.99, M = 10, \pi = 3$. Additionally, we run the popular K-means algorithm. The number of centers k is predefined as 256. Since the convergence of Kmeans algorithm depends on the application requirement (it is common to require tens of iterations, and more iterations result in more accurate result), we only report its average runtime per iteration. We can see that LSH-DDP achieves up to 70× speedup over Basic-DDP and exhibits comparable performance with Kmeans. It is also possible to lower the accuracy requirement to speedup LSH-DDP further.

10. The runtime of Basic-DDP on Activity dataset is shown '-' because we cannot finish the computation in 5 days.

TABLE 6
Runtime on EC2 (Seconds)

dataset	LSH-DDP					Basic DDP	kmeans per iter.
	job1	job2	job3	job4	job5		
USCensus	926	101	909	78	108	71,153	121.3
BigCross	1,300	156	1,353	116	134	164,172	189.4
Activity	4,319	222	4,415	158	339	-	242.8



(a) Runtime (b) Actual accuracy τ_2

Fig. 11. Runtime and actual accuracy τ_2 when varying M and π (BigCross500K).

6.5 Effect of LSH Parameters

We study the effect of LSH-DDP's parameters M and π. We run LSH-DDP on the BigCross500K data set on our local cluster of machines. We set $\mathcal{A} = 0.99$. Our offline parameter tuning (Section 5.3) approach returns $M = 5$ and $\pi = 3$. We further vary M and π. Figs. 11a and 11b report the impact of the parameters on the runtime and the accuracy metric τ_2, respectively. As shown in Fig. 11a, when $\pi = 3$, the runtime increases as M increases. However, this is not true for larger π. When $\pi = 20$, the trend actually reverses. This is because that the workload is quite skewed when M is small and π is large, leading to degraded performance. Fig. 11b shows the impact of the choice of parameters on τ_2. When M is less than 5, τ_2 is unexpectedly low and this could reduce the quality of the clustering result. On the other hand, when M is larger than 5, τ_2 is stable, achieving 99 percent accuracy for almost all cases. Taking both runtime and accuracy into consideration, we recommend to set $M = [10, 20]$ and $\pi = [3, 10]$.

7 Related Work

Clustering Techniques. Previous clustering algorithms include connectivity based clustering (e.g., hierarchical clustering [24]), centroid-based clustering (e.g., k-means [2]), distribution-based clustering (e.g., EM clustering [4]), and density-based clustering (e.g., DBSCAN [3]). As described in Section 1, Density Peaks [1] is a newly proposed clustering algorithm. DP has several distinctive advantages over previous clustering algorithms: It does not require a priori knowledge, it supports arbitrarily shaped clusters, it is deterministic, and it provides a 2D representation to visualize the input data. As a result, DP has already been employed in a wide variety of applications [5], [6], [7], [8], [9], [11]. Moreover, researchers in the AI community are

Authorized licensed use limited to: Northeastern University. Downloaded on April 06,2025 at 03:23:19 UTC from IEEE Xplore. Restrictions apply.

interested in extending DP in various aspects [25], [26], [27], [28], [29]. In this paper, we propose and evaluate LSH-DDP, an efficient distributed DP algorithm. While we focus on the original DP, we believe that it is feasible to modify our solution to support variants of DP.

MapReduce Parallelization of Sequential Algorithms. As a popular distributed programming paradigm, MapReduce has been used in parallelizing a wide range of algorithms for processing big data. This includes text processing [30], crowdsourcing [31], kNN join [32], nonnegative matrix factorization (NMF) [33], and spatial data query [34]. In this paper, we also choose MapReduce as the programming model for distributed DP algorithms.

All-Pair Computation in MapReduce. The computation of ρ and δ is related to a set of problems where computation is required for all pairs of input data elements. Kiefer et al. reduced the communication overhead of all-pair computation by using replication of set elements to enable partitioning, and by aggregating the results gathered for different copies of an element [35]. Ture et al. presented an LSH-based scalable MapReduce implementation of the sort-based sliding window algorithm for extracting pair-wise similarity [36]. In this paper, we also employ LSH in our solution. Note that Ture et al.'s work cannot be applied since ρ and δ are not similarity measurements.

Approximate Algorithms Using LSH. The LSH method was first proposed by Datar [12]. Since its introduction, LSH has been used to optimize a wide range of applications. Stupar et al. exploited LSH to answer kNN query [37]. Zhang et al. extended this work to solve kNN join problem [32]. Liu et al. employed LSH to optimize distributed graph summarization [38]. Yu et al. supported scalable content-based music retrieval through LSH [39]. Pillutla gave an approximate algorithm for distance based outlier detection using LSH [40]. We also employ LSH in our solution. As discussed in Section 1, there are several challenges in applying LSH to DP. We leverage the characteristics of DP to deal with these challenges. We present formal analysis of LSH-DDP, and show that the approximation quality and the runtime can be controlled by tuning LSH-DDP parameters.

8 Conclusion

In this paper, we present an efficient distributed algorithm LSH-DDP for Density Peaks clustering. We perform theoretical analysis of LSH-DDP, which allows users to specify the expected approximation accuracy. Compared to the naïve MapReduce implementation (Basic-DDP), LSH-DDP significantly reduces the amount of shuffled data and the computational cost, thereby achieving a factor of 1.7–70× speedups when clustering large real-world data sets. It also achieves 2× speedup over EDDPC with a very high accuracy requirement. Compared to the popular K-means clustering algorithm, LSH-DDP has comparable or better performance. In this paper, we only focus on optimizing clustering in euclidean space. By using corresponding LSH variants, LSH-DDP can be extended to support other distance metrics, such as Jaccard distance, hamming distance, cosine distance, and edit distance. In conclusion, LSH-DDP is a promising solution that makes DP algorithm feasible for clustering large real-world data sets.

Acknowledgments

This work was partially supported by the National Natural Science Foundation of China (61672141, 61528203, 61433008, 61300023), Fundamental Research Funds for the Central Universities (N141605001). The second author is partially supported by the CAS Hundred Talents program, by NSFC project No. 61572468, and by NSFC Innovation Research Group No. 61521092. S. Chen is the corresponding author.

References

[1] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *Science*, vol. 344, no. 6191, pp. 1492–1496, Jun. 2014.
[2] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. 28, no. 2, pp. 129–137, Sep. 2006.
[3] A. Dempster, N. Laird, and D. Rubin, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. 2nd Int. Conf. Knowl. Discovery Data Mining*, 1996, pp. 226–231.
[4] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *J. Roy. Statistical Soc. Series B*, vol. 39, no. 1, pp. 1–38, 1977.
[5] D. Kobak, W. Brendel, C. Constantinidis, C. E. Feierstein, A. Kepecs, Z. F. Mainen, X. Qi, R. Romo, N. Uchida, and C. K. Machens, "Demixed principal component analysis of neural population data," *eLife*, vol. 5, no. 1, pp. e10989, Apr. 2016.
[6] K. Sun, X. Geng, and L. Ji, "Exemplar component analysis: A fast band selection method for hyperspectral imagery," *IEEE Geosci. Remote Sens. Lett.*, vol. 12, no. 5, pp. 998–1002, May 2015.
[7] S. Zamuner, A. Rodriguez, F. Seno, and A. Trovato, "An efficient algorithm to perform local concerted movements of a chain molecule," *PloS One*, vol. 10, no. 3, 2015, Art. no. e0118342.
[8] D. Yu, X. Ma, Y. Tu, and L. Lai, "Both piston-like and rotational motions are present in bacterial chemoreceptor signaling," *Scientific Reports*, vol. 5, 2015, Art. no. 8640.
[9] Y. Chen, D. Lai, H. Qi, J. Wang, and J. Du, "An efficient algorithm to perform local concerted movements of a chain molecule," *PLOS ONE*, vol. 10, no. 3, pp. 0118342, Mar. 2015.
[10] N. Begum, L. Ulanova, J. Wang, and E. Keogh, "Accelerating dynamic time warping clustering with a novel admissible pruning strategy," in *Proc. Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 49–58.
[11] K. Dean, et al., "High-speed multiparameter photophysical analyses of fluorophore libraries," *Analytical Chemistry*, vol. 87, pp. 5026–5030, 2015.
[12] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. 20th Annu. Symp. Comput. Geometry*, 2004, pp. 253–262.
[13] J. S. Vitter, "Random sampling with a reservoir," *ACM Trans. Math. Softw.*, vol. 11, no. 1, pp. 37–57, 1985.
[14] W. Lu, Y. Shen, S. Chen, and B. C. Ooi, "Efficient processing of k nearest neighbor joins using MapReduce," *Proc. VLDB Endowment*, vol. 5, no. 10, pp. 1016–1027, Jun. 2012.
[15] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. 30th Annu. ACM Symp. Theory Comput.*, 1998, pp. 604–613.
[16] J. Gan, J. Feng, Q. Fang, and W. Ng, "Locality sensitive hashing scheme based on dyanmic collision counting," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2012, pp. 541–552.
[17] Y. Tao, K. Yi, C. Sheng, and P. Kalnis, "Quality and efficiency in high dimensional nearest neighbor search," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 563–576.
[18] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe LSH: Efficient indexing for highdimensional similarity search," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 950–961.
[19] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proc. 25th Int. Conf. Very Large Data Bases*, 1999, pp. 518–529.
[20] Y. Liu, J. Cui, Z. Huang, H. Li, and H. T. Shen, "SK-LSH: An efficient index structure for approximate nearest neighbor search," *Proc. VLDB Endowment*, vol. 7, pp. 745–756, 2014.

Authorized licensed use limited to: Northeastern University. Downloaded on April 06,2025 at 03:23:19 UTC from IEEE Xplore. Restrictions apply.

[21] V. Zolotarev, "One-dimensional stable distributions," *Translations Math. Monographs*, vol. 65, pp. 270–277, 1986.
[22] M. Rojas, S. A. Santos, and D. C. Sorensen, "Algorithm 873: LSTRS: MATLAB software for large-scale trust-region subproblems and regularization," *ACM Trans. Math. Softw.*, vol. 34, no. 2, pp. 11:1–11:28, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1326548.1326553
[23] S. Gong and Y. Zhang, "EDDPC: An efficient distributed density peaks clustering algorithm," *J. Comput. Res. Develop.*, vol. 53, no. 6, pp. 1400–1409, Jun. 2016.
[24] L. Kaufman and P. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. New York, NY, USA: Wiley, 1990.
[25] J. Xu and G. Wang. "Leading tree in DP_CLUS and its impact on building hierarchies," 2015. [Online]. Available: http://arxiv.org/abs/1506.03879
[26] Z. Xie, L. Jiang, T. Ye, and X. Li, "A synthetic minority oversampling method based on local densities in low-dimensional space for imbalanced learning," *Database Systems for Advanced Applications*. Berlin, Germany: Springer, 2015.
[27] A. Decelle and F. Ricci-Tersenghi, "Solving the inverse ising problem by mean-field methods in a clustered phase space with many states," arXiv:1501.03034, 2015.
[28] S. Wang, D. Wang, C. Li, and Y. Li, "Comment on clustering by fast search and find of density peaks," arXiv: 1501.04267, 2015.
[29] W. Zhang and J. Li, "Extended fast search clustering algorithm: Widely density clusters, no density peaks," arXiv: 1505.05610, 2015.
[30] J. Lin and C. Dyer, *Data-Intensive Text Processing with MapReduce*. San Mateo, CA, USA: Morgan and Claypool, 2010.
[31] P. Langhans, C. Wieser, and F. Bry, "Crowdsourcing mapreduce: Jsmapreduce," in *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 253–256.
[32] C. Zhang, F. Li, and J. Jestes, "Efficient parallel kNN joins for large data in MapReduce," in *Proc. 15th Int. Conf. Extending Database Technol.*, 2012, pp. 38–49.
[33] C. Liu, H.-C. Yang, J. Fan, L.-W. He, and Y.-M. Wang, "Distributed nonnegative matrix factorization for web-scale dyadic data analysis on MapReduce," in *Proc. 19th Int. Conf. World wide web*, 2010, pp. 681–690.
[34] A. Eldawy, M. F. Mokbel, S. Alharthi, A. Alzaidy, K. Tarek, and S. Ghani, "SHAHED: A MapReduce-based system for querying and visualizing spatio-temporal satellite data," in *Proc. IEEE 31st Int. Conf. Data Eng.*, 2015, pp. 1585–1596.
[35] T. Kiefer, P. B. Volk, and W. Lehner, "Pairwise element computation with MapReduce," in *Proc. 19th ACM Int. Symp. High Performance Distrib. Comput.*, 2010, pp. 826–833.
[36] F. Ture, T. Elsayed, and J. Lin, "No free lunch: Brute force versus locality-sensitive hashing for cross-lingual pairwise similarity," in *Proc. 34th Int. ACM SIGIR Conf. Res. Development Inf. Retrieval*, 2011, pp. 943–952.
[37] A. Stupar, S. Michel, and R. Schenkel, "RankReduce processing k-nearest neighbor queries on top of MapReduce," in *Proc. 8th Workshop Large-Scale Distrib. Syst. Inf. Retrieval*, pp. 13–18, 2010.
[38] X. Liu, Y. Tian, Q. He, W.-C. Lee, and J. McPherson, "Distributed graph summarization," in *Proc. 23rd ACM Int. Conf. Inf. Knowl. Manage.*, 2014, pp. 799–808.
[39] Y. Yu, R. Zimmermann, Y. Wang, and V. Oria, "Scalable content-based music retrieval using chord progression histogram and tree-structure LSH," *IEEE Trans. Multimedia*, vol. 15, no. 8, pp. 1969–1981, Dec. 2013.
[40] M. R. Pillutla, N. Raval, P. Bansal, K. Srinathan, and C. V. Jawahar, "LSH based outlier detection and its application in distributed setting," in *Proc. 20th ACM Int. Conf. Inf. Knowl. Manage.*, 2011, pp. 2289–2292.

Yanfeng Zhang received the PhD degree in computer science from Northeastern University, China, in 2012. He is currently an associate professor with Northeastern University, China. His research consists of distributed systems and big data processing. He has published many papers in the above areas. His paper in SoCC 2011 was honored with "Paper of Distinction".

Shimin Chen received the PhD degree from Carnegie Mellon University, in 2005. He is a professor at ICT CAS. He worked as a researcher, senior researcher, and research manager at Intel Labs , Carnegie Mellon University, and HP Labs before joining ICT. His research interests include data management systems, computer architecture, and big data processing.

Ge Yu received the PhD degree in computer science from the Kyushu University of Japan, in 1996. He is now a professor with Northeastern University, China. His current research interests include distributed and parallel database, OLAP and data warehousing, data integration, graph data management, etc. He has published more than 200 papers in refereed journals and conferences. He is a member of the IEEE, the ACM and the CCF.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.

Authorized licensed use limited to: Northeastern University. Downloaded on April 06,2025 at 03:23:19 UTC from IEEE Xplore. Restrictions apply.