Parallel Massive Clustering of Discrete Distributions

YU ZHANG, JAMES Z. WANG and JIA LI, The Pennsylvania State University

The trend of analyzing big data in artificial intelligence demands highly-scalable machine learning algorithms, among which clustering is a fundamental and arguably the most widely applied method. To extend the applications of regular vector-based clustering algorithms, the Discrete Distribution (D2) clustering algorithm has been developed, aiming at clustering data represented by bags of weighted vectors which are well adopted data signatures in many emerging information retrieval and multimedia learning applications. However, the high computational complexity of D2-clustering limits its impact in solving massive learning problems. Here we present the parallel D2-clustering (PD2-clustering) algorithm with substantially improved scalability. We developed a hierarchical multi-pass algorithm structure for parallel computing in order to achieve a balance between the individual-node computation and the integration process of the algorithm. Experiments and extensive comparisons between PD2-clustering and other clustering algorithms are conducted on synthetic datasets. The results show that the proposed parallel algorithm achieves significant speed-up with minor accuracy loss. We apply PD2-clustering to image concept learning. In addition, by extending D2-clustering to symbolic data, we apply PD2-clustering to protein sequence clustering. For both applications, we demonstrate the high competitiveness of our new algorithm in comparison with other state-of-the-art methods.

Categories and Subject Descriptors: I.5.3 [**Computing Methodologies**]: Clustering—*Algorithms*; D.1.3 [**Software**]: Concurrent Programming—*Distributed programming*

Additional Key Words and Phrases: Discrete distribution, clustering, parallel computing, large-scale learning, image annotation, protein clustering

ACM Reference Format:
Yu Zhang, James Z. Wang, and Jia Li. 2014. Parallel Massive Clustering of Discrete Distributions. *ACM Trans. Multimedia Comput. Commun. Appl.* 1, 1, Article 1 (January 2015), 23 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Clustering is a fundamental unsupervised learning methodology for data mining and machine learning. Additionally, clustering can be used in supervised learning for building non-parametric models [Beecks et al. 2011]. The K-means algorithm [MacQueen 1967] is one of the most widely used clustering algorithms because of its simple yet generally accepted objective function for optimization. Though several variants of K-means have evolved recently (e.g., spherical K-means [Dhillon and Modha 2001] and clustering with Bregman divergences [Banerjee et al. 2005]), a fundamental limitation of K-means and those variants is that they only apply to the Euclidean space.

In many emerging applications, however, the data to be processed are not vectors. In particular, bags of weighted vectors (e.g., bags of words, bags of visual words, sparsely

This work is supported by the National Science Foundation Grant No. 1027854, 0347148, and 0936948.
Author's address: Yu Zhang, James Z. Wang, College of Information Sciences and Technology, Penn State University, University Park, PA 16802; email: {yzz123,jwang}@ist.psu.edu; Jia Li, Department of Statistics, Penn State University, University Park, PA 16802; email: jiali@stat.psu.edu.

DOI:http://dx.doi.org/10.1145/0000000.0000000

represented histograms), which can be formulated as discrete distributions with finite but arbitrary supports, are often used as object descriptors. The discrete distribution clustering algorithm, also known as the *D2-clustering* algorithm [Li and Wang 2008], tackles such type of data in the same spirit of K-means. Specifically, it groups discrete distributions into clusters and calculates a centroid for each cluster by minimizing the sum of squared distances between each object and its nearest centroid. The clustering result (i.e., cluster labels and centroids) is applied to the mixture modeling of image concepts in a real-time image annotation system named ALIPR [Li and Wang 2008].

The *Mallows distance*[1] is adopted in D2-clustering as the distance measure of discrete distributions. The distance is defined by solving the *transportation problem* [Monge 1781; Kantorovich 1942], which finds an optical matching between two discrete distributions. Compared to other metrics for discrete distributions such as $\mathscr{L}^1$ distance [Batu et al. 2013] and Kullback-Leibler divergence [Cover and Thomas 2012] which treat discrete distributions as regular histograms and only compare the aligned bins, the Mallows distance does not require the supports to be aligned and takes into account their locations in computation. Therefore it can more effectively measure the distance between discrete distributions with arbitrary supports, which is common in the bags-of-words (BoW) representation. In addition to image retrieval and annotation, the Mallows distance is also adopted in multiple machine learning applications such as video classification and retrieval [Xu and Chang 2008], sequence categorization [Pond et al. 2010], and document retrieval [Wan 2007], where objects are represented by discrete distributions. As a fundamental tool to group discrete distributions with Mallows distance, D2-clustering can be highly valuable in these areas.

A major concern in applying D2-clustering is its scalability. Though interlacing the update of cluster labels and centroids in the same manner as K-means, D2-clustering updates each cluster centroid by solving a linear program involving all the objects in the cluster. Because the linear programming (LP) problem has a high-order polynomial complexity, the computational cost increases polynomially with the sample size (assuming the number of clusters is roughly fixed). In our pilot study, we test the D2-clustering algorithm on a 2 GHz single-core CPU by clustering a set of image BoW descriptors. It takes several minutes to complete a clustering of 80 images, which is the number reported in [Li and Wang 2008] for modeling an image concept. However, it demands more than four days to cluster 1,000 images, which makes the algorithm impractical on large datasets.

The clustering problems researchers tackle today are often of very large scale. For instance, we are seeking to perform learning and retrieval on images from the Web containing millions of pictures annotated by one common word. This leads to demanding clustering tasks even if we perform learning only on a tiny fraction of the Web. Adopting bags of weighted vectors as descriptors, we can encounter the same issue in other applications such as video, biological sequence, and text clustering, where the amount of data can be enormous. In such cases, the original D2-clustering algorithm is computationally impractical. Given the application potential, it is of great interest to find an efficient way to perform large-scale discrete distribution clustering.

In this paper, we introduce a parallelized version of the original D2-clustering algorithm, namely the *PD2-clustering* algorithm. To distinguish the proposed method with the original D2-clustering algorithm, we hereafter refer to the proposed algorithm as the *parallel algorithm*, and the original algorithm as the *sequential algorithm*. Because of the intrinsic nonlinearity and complexity of D2-clustering, its parallelization is not trivial and requires a properly designed algorithm structure that guarantees

[1]Also known as the Kantorovich-Wasserstein metric and the Earth Mover's Distance (EMD) in different scenarios.

both high efficiency and little approximation. The following research questions (RQs) are answered in the design and analysis of the PD2-clustering algorithm:

RQ 1: How to parallelize the non-linear clustering and how the scalability improves?
RQ 2: How to reduce accuracy loss in the parallelization design?
RQ 3: What are the advantages of PD2-clustering and where can it be applied?

It is worth mentioning that besides parallelization, there are several other strategies that can be applied in large-scale discrete distribution clustering tasks. Some common approaches are:

(a) Better implementation: adopting well optimized commercialized LP solvers to accelerate the centroid update operation in sequential D2-clustering;
(b) Simpler data signature and metric: performing vector quantization to transform BoW descriptors to vectors, then using conventional vector clustering algorithms to bypass D2-clustering.
(c) Connectivity-based clustering: computing the pair-wise (Mallows) distance between every two discrete distributions and applying connectivity-based clustering algorithms such as agglomerative clustering and spectral clustering, which can be easily parallelized.

Compared with the above-mentioned approaches, the proposed PD2-clustering algorithm has three advantages: first, it is a structure but not implementation level optimization of the sequential algorithm, therefore essentially lowers the time-complexity; second, data entries are clustered based on their discrete distribution representations without being converted to other types of less descriptive representations or pair-wise distances; third, cluster centroids, which are also discrete distributions, are obtained after the clustering, i.e., both the cluster assignments and the cluster representatives are obtained. As a result, PD2-clustering performs well in terms of efficiency, performance, and application potential. Experiments comparing PD2-clustering with other methods demonstrate these advantages.

The remainder of the paper is organized as follows. Section 2 introduces the Mallows distance and D2-clustering algorithm, which are the basics of this work. A review on existing parallel clustering algorithms is given in Section 3. Then we introduce the proposed parallel D2-clustering algorithm in Section 4, followed by experiments on synthetic and real-world large-scale datasets in Section 5. Finally, we answer the three research questions presented above and conclude the paper in Section 6.

2. PRELIMINARIES

2.1. Bag-of-Words Model and Discrete Distribution

The Bag-of-Words (BoW) model is an efficient representation broadly adopted in natural language processing and information retrieval [Zheng and Gao 2008; Sang and Xu 2011]. Given a dictionary $\mathscr{D}$, it summarizes a document by the occurrence of each word defined by $\mathscr{D}$. For example, if words $v^{(1)}, v^{(2)}, \ldots, v^{(t)} \in \mathscr{D}$ occur in a document with counts $f^{(1)}, f^{(2)}, \ldots, f^{(t)}$, the BoW descriptor of the document is the concatenation of all the word-frequency tuple $V = \{(v^{(1)}, f^{(1)}), (v^{(2)}, f^{(2)}), \ldots, (v^{(t)}, f^{(t)})\}$. The BoW model can be generalized to any type of "documents" (e.g., web pages, images, biological sequences, etc.) where the "words" are the basic elements in the "documents" (e.g., linguistic words, visual objects, and nucleotides).

Similar to the BoW model, a discrete distribution is formulated as a bag of weighted objects, $V = \{(v^{(1)}, p^{(1)}), (v^{(2)}, p^{(2)}), \ldots, (v^{(t)}, p^{(t)})\}$, in which the probability of a random variable v is non-zero only at a finite number of locations $v^{(1)}, v^{(2)}, \ldots, v^{(t)}$ in the sample space, and $\sum_{i=1}^{t} p^{(i)} = 1$. A document can be represented by a discrete distribution by modeling its inclusion of certain elements as a random variable v and the probability $p^{(i)} = P(v = v^{(i)})$ as $v^{(i)}$'s percentage in the document. Compared with the BoW model, the discrete distribution model adopts a probability measure instead of a counting measure for each element. In addition, a dictionary $\mathscr{D}$ with finite cardinality is not necessary for a discrete distribution. It only requires the number of supports to be finite and the support v can take any arbitrary value (i.e., the random variable's sample space can be continuous).

2.2. Transportation Metrics

Whether or not $\mathscr{D}$ exists, both BoWs and discrete distributions are sparse representations because the cardinality of the sample space (vocabulary size) is typically much larger than the number of supports in both representations. Even though we can regard a BoW as a vector in a space spanned by $\mathscr{D}$, it is inefficient to compare BoWs by a general vector space measurement. We therefore introduce the transportation metric to measure BoWs and discrete distributions [Levina and Bickel 2001].

Given two discrete sets $\{v_i^{(1)}, v_i^{(2)}, \ldots, v_i^{(t_i)}\}$ and $\{v_j^{(1)}, v_j^{(2)}, \ldots, v_j^{(t_j)}\}$, both in a Radon space with a Radon measure [2] μ, the transportation problem is formulated as:

$$\begin{aligned}
&\min_{w_{a,b}} \sum_{a=1}^{t_i} \sum_{b=1}^{t_j} w_{a,b} C(v_i^{(a)}, v_j^{(b)}) \\
\text{subject to:}\quad & \\
& w_{a,b} > 0, \forall a \in \{1, \ldots, t_i\}, \forall b \in \{1, \ldots, t_j\}, \\
& \sum_{a=1}^{t_i} w_{a,b} \le \mu(v_j^{(b)}), \forall b \in \{1, \ldots, t_j\}, \\
& \sum_{b=1}^{t_j} w_{a,b} \le \mu(v_i^{(a)}), \forall a \in \{1, \ldots, t_i\}, \\
& \sum_{a=1}^{t_i} \sum_{b=1}^{t_j} w_{a,b} = \min\left(\sum_{a=1}^{t_i} \mu(v_i^{(a)}), \sum_{b=1}^{t_j} \mu(v_j^{(b)})\right) .
\end{aligned} \tag{1}$$

where $w_{a,b}$ is the flow (or match) between elements $v_i^{(a)}$ and $v_j^{(b)}$, and $C(\cdot, \cdot)$ is a general cost function that is Borel-measurable. The optimization is a linear program, which can be solved by the Simplex algorithm [Murty 1983] in polynomial time (with respect to the number of variables $w_{a,b}$).

The transportation problem solves a minimal weighted cost for transforming one set to another. Normalizing the minimal cost by the total flow, we obtain the transportation metric:

$$T(V_i, V_j) = \frac{\min_{w_{a,b}} \sum_{a=1}^{t_i} \sum_{b=1}^{t_j} w_{a,b} C(v_i^{(a)}, v_j^{(b)})}{\sum_{a=1}^{t_i} \sum_{b=1}^{t_j} w_{a,b}}, \quad \text{subject to the constraints in Equation (1).} \tag{2}$$

When $\mu(\cdot)$ is the counting measure and the transportation cost $C(v_i^{(a)}, v_j^{(b)})$ is the distance between $v_i^{(a)}$ and $v_j^{(b)}$, Equation (2) defines the Earth Mover's Distance between two BoWs. For discrete distributions, $\mu(\cdot)$ is the probability measure, i.e.,

[2]Both the counting measure and probability measure on a discrete set are Radon measures.

$\sum_{a=1}^{t_i} \mu(v_i^{(a)}) = \sum_{b=1}^{t_j} \mu(v_j^{(b)}) = \sum_{a=1}^{t_i} \sum_{b=1}^{t_j} w_{a,b} = 1$, all constrains in Equation (1) take the equal signs. The Mallows distance between two discrete distributions is formulated as:

$$D(V_i, V_j) = \min_{w_{a,b}} \sum_{a=1}^{t_i} \sum_{b=1}^{t_j} \left(w_{a,b} \| v_i^{(a)} - v_j^{(b)} \|^p \right)^{\frac{1}{p}}, \qquad (3)$$

subject to the constraints in Equation (1),

where the cost function $\| \cdot \|$ is typically the $\mathscr{L}^1$ or $\mathscr{L}^2$ vector norm. $p \geq 1$ is the degree of the Mallows distance and $p = 1$ and $p = 2$ are most adopted. Without specifying, we use the $\mathscr{L}^2$ norm as the cost and $p = 2$ for the Mallows distance in the paper.

Because the representations of BoWs and discrete distributions, as well as the corresponding metrics (EMD or Mallows distance), are defined in an identical way, we hereafter only refer to discrete distributions and Mallows distance in the paper. Our approach works in both cases.

2.3. D2-clustering Algorithm

D2-clustering is introduced as the K-means' counterpart in the discrete distribution space with the Mallows distance as the metric. It groups discrete distributions into clusters and finds the centroid of each cluster. Operating in the same spirit as K-means clustering, it also aims at minimizing the total within-cluster dispersion, which is the sum of squared Mallows distances from each data point to its closest cluster centroid. Denote each discrete distribution as $V_i = \{(v_i^{(1)}, p_i^{(1)}), \ldots, (v_i^{(t_i)}, p_i^{(t_i)})\}$ and assume there are N discrete distributions $V_1, \ldots, V_N$ to be clustered into k groups with centroids $Z_j = \{(z_j^{(1)}, q_j^{(1)}), \ldots, (z_j^{(s_j)}, q_j^{(s_j)})\}, j = 1, \ldots, k$, the objective is formulated as:

$$\psi = \sum_{j=1}^{k} \sum_{i:L(i)=j} D^2(Z_j, V_i) \ , \qquad (4)$$

where $D(\cdot, \cdot)$ denotes the Mallows distance between two discrete distributions, and $L(\cdot)$ is the cluster assignment function, $L(i) = j$ if the data point V_i belongs to the cluster centered at Z_j.

D2-clustering algorithm is analogous to K-means in that it iterates the update of the partition and the update of the centroids. The algorithm iteratively performs the following two stages until the objective defined in Equation (4) converges.

(1) Keep cluster centroids fixed and update the partition by assigning each data point with the label of its closest centroid:

$$L(i) = \operatorname*{argmin}_{j \in \{1,\ldots,k\}} D(Z_j, V_i) \ . \qquad (5)$$

(2) Keep $L(\cdot)$ fixed and solve the centroid that minimizes within-cluster dispersion for each cluster:

$$Z_j = \operatorname*{argmin}_{Z} \sum_{i:L(i)=j} D^2(Z, V_i) \ . \qquad (6)$$

Linear programs are involved in both stages due to the introduction of the Mallows distance. However, it is the minimization in Equation (6) that makes D2-clustering substantially more complex than K-means. We expand the Mallows distance $D(\cdot, \cdot)$ in Equation (6) and rewrite the minimization as:

$$\min_{\theta} \sum_{i:L(i)=j} \sum_{a=1}^{s_j} \sum_{b=1}^{t_i} w_{a,b}^{(i)} \| z_j^{(a)} - v_i^{(b)} \|^2 \ , \qquad (7)$$

where $\theta = \{z_j^{(a)}, q_j^{(a)}, w_{a,b}^{(i)} : L(i) = j, a = 1, \ldots, s_j, b = 1, \ldots t_i\}$ is the set containing all the variables in the optimization. To solve the complex optimization, D2-clustering updates $z_j^{(a)}$'s and $\{q_j^{(a)}, w_{a,b}^{(i)}\}$'s iteratively until the solution is stable.

It first keeps $z_j^{(a)}$'s fixed[3] and updates $\{q_j^{(a)}, w_{a,b}^{(i)}\}$'s in Equation (7). The minimization reduces to a linear program with constraints:

$$\begin{aligned}
&\sum_{b=1}^{t_i} w_{a,b}^{(i)} = q_j^{(a)}, \forall (i,a) \text{ s.t. } L(i) = j, 1 \le a \le s_j \ , \\
&\sum_{a=1}^{s_j} w_{a,b}^{(i)} = p_i^{(b)}, \forall (i,b) \text{ s.t. } L(i) = j, 1 \le b \le t_i \ , \\
&\sum_{a=1}^{s_j} q_j^{(a)} = 1, q_j^{(a)} \ge 0, \text{and } w_{a,b}^{(i)} \ge 0, \forall i, a, b \ .
\end{aligned} \tag{8}$$

Then $z_j^{(a)}$'s are updated as:

$$z_j^{(a)} = \frac{\sum_{i:L(i)=j} \sum_{b=1}^{t_i} w_{a,b}^{(i)} v_i^{(b)}}{\sum_{i:L(i)=j} \sum_{b=1}^{t_i} w_{a,b}^{(i)}}, a = 1, \ldots, s_j. \tag{9}$$

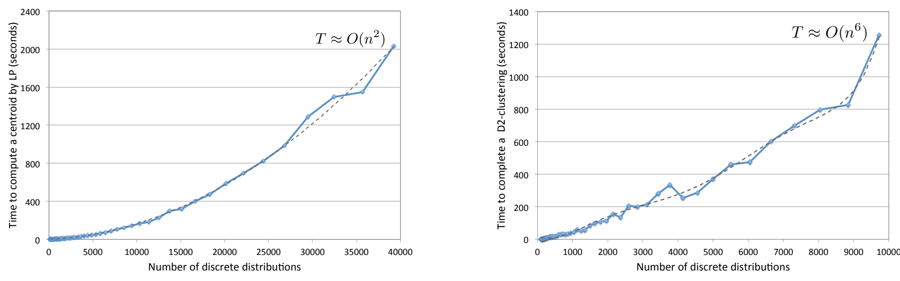
The linear program involves $s_j + s_j \sum_{i:L(i)=j} t_i$ variables and contains $1 + \sum_{i:L(i)=j} (t_i + s_j)$ constrains. With the number of supports s_j and t_i relatively constant, the scale of the linear program is proportional to the number of samples in cluster j, $|i : L(i) = j|$. Due to the intrinsic polynomial complexity of the LP problem, the centroid update stage on large datasets is slow for D2-clustering. Fig. 1 shows the results of the scalability tests of D2-clustering, as well as the embedded centroid update optimization defined by Equation (7) on synthetic datasets with different sizes. We adopt the IBM ILOG CPLEX Optimization Studio (CPLEX), one of the most popular and powerful commercial optimizers, to solve the LPs in D2-clustering. Even though CPLEX adopts certain pre-solve and constraint-reduction mechanisms to simplify a LP and accelerate the optimization before actually starting the Simplex algorithm, the curve in Fig. 1(a) still shows a polynomial time complexity to complete a single round of centroid update. When the centroid update operation is embedded in the D2-clustering algorithm and invoked polynomial times, the overall run-time of the clustering is of a higher ordered polynomial complexity (see Fig. 1(b)). In fact, the D2-clustering computation cannot finish within a practical time if we further increase the size of the dataset. Detailed run-time analysis is presented in Appendix A.

The computational intensiveness of D2-clustering limits its usages to only relatively small scale problems, in terms of number of data points per training class. With emerging demands to extend the algorithm to large-scale multimedia datasets, e.g., online image datasets, video resources, and biological data, we exploit parallel processing to overcome the inadequate scalability of D2-clustering.

3. RELATED WORK

The parallel computing technique is widely applied to extend existing clustering algorithms to large-scale problems, especially with the growing trend and power of cloud

[3]Before the centroid update iteration, all support vectors from the discrete distributions in the cluster, $\{v_i^{(b)} : v_i^{(b)} \in V_i, L(i) = j\}$, are grouped to s_j clusters by K-means, and the s_j centroid vectors are used as the initial values of the supports for Z_j,



(a) Centroid update time vs. scale (b) D2-clustering time vs. scale

Fig. 1 . Run-time of the sequential D2-clustering algorithm and the embedded centroid update operation which is invoked iteratively. (a) Given different numbers of discrete distributions, the optimizations defined in Equation (7) are solved to find a single centroid. The run-time is approximately quadratic to the number of data points. (b) Iteratively invoking the centroid update operation in (a), a D2-clustering is performed to group the dataset into 10 clusters. The overall run-time is polynomial with an approximate degree of 6. The solid curves plot the average time of 5 runs and the dashed curves are approximate polynomial fitting results of the run-time vs. number of data points. In each run the dataset is randomly generated and each discrete distribution contains 5 supports. The LPs are solved by functions in IBM CPLEX and the CPU's clock frequency is 2.7 GHz.

computing nowadays. Several parallel clustering algorithms have been developed in different ways based on the original clustering approaches being parallelized. In general, a parallel clustering algorithm adopts the divide-and-conquer strategy to first divide the large-scale clustering problem into sub-problems and then merge the results of these sub-problems to a global one. Different parallel algorithms vary in the manners they divide the original problem and conquer the sub-problems. Table I compares the run-time of several popular parallel clustering algorithms and the proposed PD2-clustering algorithm.

The K-means algorithm is the most widely adopted clustering method for vectors. Given an initial set of k cluster centroids, it iteratively updates the cluster assignments and cluster centroids until the solution is stable. Within each iteration, the cluster assignments and cluster centroids are alternatively updated by fixing one set of variables and optimizing the rest towards the object, which is the sum of the squared Euclidean distance from each data point to its corresponding cluster centroid (i.e., total within-cluster dispersions). K-means can be easily parallelized because both the cluster assignment and centroid update are straightforwardly separable. In particular, the optimal centroid that minimizes the within-cluster dispersion of a given cluster of data is simply the arithmetic mean of them, which indicates that the global centroids can be easily computed by a weighted average of the local means. Zhao et al. [2009] introduced a Map-Reduce implementation of the parallel K-means (PKMeans) algorithm. The data points are randomly divided into chunks, each of which is handled by a mapper. For each point, the corresponding mapper computes its distances to all the (global) cluster centroids, and assigns it to the cluster it is closest to. Local centroids within each chunk are computed and transmitted to different reducers based on their cluster labels. Each reducer only receives local centroids with a same label from different mappers, and merges them to a global cluster centroid by taking the weighted average of them. Such Map-Reduce process is iteratively invoked until convergence. Gonina et al. [2014] used a similar strategy to parallelize the clustering where clusters are modeled by Gaussian Mixture Models (GMMs). The Expectation-Maximization (EM) iteration is performed locally in parallel before nearby local GMMs are merged.

When only the data grouping is needed and cluster centroids are not necessary, the hierarchical clustering and spectral clustering algorithms can be applied by analyzing the connectivity or pair-wise distances among data points. The general idea to parallelize hierarchical clustering [Olson 1995; Dahlhaus 2000] is to compute and update the distance matrix between data points and clusters in parallel during the construction of the clustering dendrogram. Every parallel processor has a local copy of all the data and the dendrogram is broadcast after every updating. The parallelization splits the distance computation to M parallel processors, therefore can reduce the overall clustering run time from $O(N^2 \log N)$ to $O(N^2 \log N/M)$.

Chen et al. [2011] presented the parallel spectral clustering (PSC) algorithm implemented by the Message Passing Interface (MPI). As other connectivity-based parallel clustering algorithms, it first computes the distance matrix in parallel, where the whole dataset is evenly split and dispatched to M processors. It then employs a parallel eigensolver to perform the eigen-decomposition of the corresponding Laplacian matrix, and adopts the PKMeans algorithm to group the data points' projections in the eigenspace. An M-time acceleration is acquired in each step.

Ferreira Cordeiro et al. [2011] introduced a "Best of both worlds" prototype to perform parallel subspace clustering. Two approaches, parallel subspace clustering (ParC) and Sample-and-Ignore (SnI), are utilized. They adopt the divide-and-conquer and random sampling strategies respectively to accelerate the clustering of large datasets. Cost functions for both approaches are given. For a certain clustering task, the algorithm selects the clustering approach (ParC or SnI) with less cost based on the size of data and the hardware parameters. The approach can efficiently perform subspace clustering on up to 1.4 billion high-dimensional data points by MapReduce.

Although the objectives and implementations of the above approaches are different, they all adopt the divide-and-conquer strategy. The dividing (mapper) operation, as well as the parallel tasks, are heavily loaded, but shared among processors. The merging (reducer) operation gathers all the local clusters, combines overlapping (or nearby) clusters, or updates global centroids from local centroids. Because the number of clusters are much smaller than the number of data points, the merging operation is typically simple and fast to complete and performed by a single processor.

The parallelization of D2-clustering is more complex than all the above parallel clustering algorithms. Because both the data points and cluster centroids are discrete distributions, LPs are involved in both the "divide" and "conquer" stages of the parallel algorithm. In particular, the merging of local centroids is complex. We therefore seek a multi-pass structure that iteratively merges the local results. The clustering strategy is similar to that used in the stream clustering algorithm [Guha et al. 2003], where data arrive sequentially in a stream and the total amount is large and unknown. In stream clustering, data are chunked by their arrival order and clustered locally. Then the local clusters are merged in a hierarchical way, in which the cluster centroids are chunked, clustered, and iteratively merged. Stream clustering is not performed in parallel because the clustering of different chunks are invoked sequentially as new data arrive. It does little job in dividing the data, which are simply chunked by their arrival or storage order. Instead, it adopts a sophisticated merging approach in a hierarchical structure. The proposed PD2-clustering algorithm constructs a similar hierarchy, but is more flexible in the data chunking, and cluster merging operations because the data are not arriving in a stream. In addition, the local clustering tasks are performed in parallel. The details are introduced in Section 4.

4. PARALLEL D2-CLUSTERING ALGORITHM

4.1. Algorithm Structure

Fig. 2(a) illustrates the general workflow of a parallel clustering algorithm. Data points are first divided into chunks and distributed to different processors. Each processor

Table I Run-time Comparison between PD2-clustering and Other Parallel Clustering Algorithms

Algorithm	Before parallelization	After parallelization
D2-clustering [a]	$O(f_{D2}(N)) = O(N^{\alpha})$	$O(\frac{1}{M}(N^{\beta} + f_{D2}(\tau)\frac{N}{\tau}))$
K-means clustering [b]	$O(f_{KM}(N)) = O(Nkt)$	$O(\frac{1}{M}f_{KM}(N))$
Spectral clustering [c]	$O(f_{SC}(N)) = O(N^2)$	$O(\frac{1}{M}f_{SC}(N))$
Hierarchical clustering [d]	$O(f_{HC}(N)) = O(N^2 \log N)$	$O(\frac{1}{M}f_{HC}(N))$

Note: Denote N as the size of dataset, and M as the number of processors.
[a] $\alpha \gg \beta > 1$, τ is the chunk size. See Appendix A for the details.
[b] k is the desired cluster number, and t is the number of iteration.
[c] The Noyström method is applied to approximate the spectral matrix in both the sequential and parallel versions. Computation of pair-wise distances leads the run-time.
[d] Complete link is adopted as the graph metric.

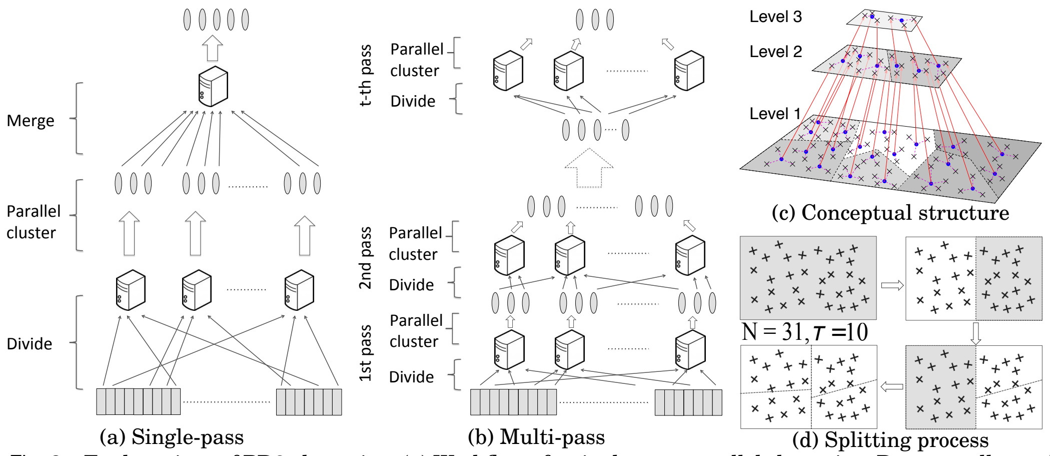


Fig. 2 . Explanations of PD2-clustering. (a) Workflow of a single-pass parallel clustering. Data are allocated to different processors for parallel clustering. The "merge" stage combines local clusters at once. (b) Workflow of a multi-pass parallel clustering. Local clusters are gradually merged by further clustering local centroids level by level. (c) The conceptual hierarchical structure of the multi-pass clustering. Crosses represent data points and dots represent centroids. Points are clustered locally in different chunks. Centroids in certain level are treated as data points in its next level and further clustered. (d) An illustration of a binary splitting process in the "divide" phase. The largest (shaded) segment is split until all segments are below size τ.

clusters the parallel data chunks it receives separately by a certain sequential plug-in algorithm, specifically D2-clustering in our case. The parallel clustering results are then combined to a global solution in the "merge" stage. The dividing operation, namely the mapper function in MapReduce, computes an allocation for each data point. As a preparation step of the real parallel computing, it is supposed to be fast and usually an approximate partition of the dataset. As a result, the local clusters obtained from the parallel tasks are overlapped, and some clusters need to be combined in the "merge" stage. The merging is performed by a single processor which gathers the local results and groups overlapping clusters together.

It should be emphasized that the centroids generated by D2-clustering, as representatives of clusters of discrete distributions, are also discrete distributions. The merging operation of the local centroids are therefore complex because of the involvement of LP. To achieve a low overall run-time, we want to keep both the local clustering tasks and the combination of local results at a small scale, which cannot be satisfied simultaneously when the size of the overall data is large. Because the merging operation is in fact a clustering process of the local centroids per se, we propose a multi-pass workflow that invokes the single-pass parallel clustering repeatedly. When the number of local cluster centroids are large, the algorithm feeds them to another pass of parallel clustering instead of attempting to complete the merging at once. The same operation is performed iteratively, until certain stopping criteria (to be discussed later) are satisfied. As a result, the clustering is done hierarchically and within each level of the hierarchy, the scales of LPs are not large.

Fig. 2(b) illustrates the process of multi-pass parallel D2-clustering. Compared to the single-pass way demonstrated in Fig. 2(a) which merges the local centroids from the parallel tasks by a single processor, the multi-pass approach invokes another pass of parallel clustering to merge the local centroids, and repeatedly applies the divide-and-conquer strategy until the stopping criteria are satisfied. Consequently it avoids solving a large LP in the merging process. In other words, the multi-pass structure gradually "conquers" the local clustering results in parallel processors rather than merging them with a single processor immediately.

Conceptually the multi-pass clustering forms a hierarchical structure demonstrated in Fig. 2(c). Apparently the hierarchy always converges because each level contains less number of objects than its previous level. The algorithm uses the user-defined shrinking factor R to control the converging speed of the hierarchy and the chunk size τ to control the scale of local clustering tasks. For each local clustering task involving N' data points ($N' \leq \tau$), the target cluster number is ${}^{N'}\!/_{R}$. As a result the total number of data to be cluster in the next pass shrinks R times (see Section 5.2 for the selection of R and τ). Such a structure achieves an overall run-time

$$T(N) = \frac{1}{M}\Theta(N^{\beta} + \tau^{\alpha-1}N) \;, \tag{10}$$

where N is the number of data points, M is the number of processors, and $\alpha \gg \beta > 0$ are constants (see Appendix A for detailed analysis). It is also shown in Appendix C that PD2-clustering algorithm is a $(1+\varepsilon)$-approximation of D2-clustering where

$$1 + \varepsilon = N^{\kappa+\lambda}\tau^{-\lambda}R^{-\kappa}\sum_{\gamma=0}^{l-1} 2^{l-\gamma}R^{-\gamma(\kappa+\lambda)} \tag{11}$$

(k is the cluster number, $l = \lceil \log_R \frac{N}{k} \rceil$ is the number of passes, $\kappa < 0$ and $\lambda > 0$).

Though initially developed for the PD2-clustering algorithm, this framework can be in fact applied to the parallelization of other clustering algorithms where cluster prototypes are given just by substituting the plug-in clustering method in the parallel tasks. Such way of parallelization is especially beneficial for algorithms where the merging of parallel results is complex.

4.2. Data Segmentation Method

We now describe the initial data segmentation method in the "divide" phase. In this step, we aim to partition the dataset into groups to be clustered by parallel processors. To keep the sub-tasks simple, we want all the groups to be smaller than τ. In addition, we hope data in each segment to be tight for less approximation (see Appendix C).

The partitioning process is a series of greedy binary splitting. It is similar in spirit as the initialization step of the LBG algorithm proposed by Linde et al. [1980], an early instance of K-means in signal processing. The whole dataset is iteratively split into partitions until a stopping criterion is satisfied. Different from the initialization of the LBG algorithm, which splits every segment into two parts, doubling the total number of segments in every iteration, our approach only splits one segment containing the most number of data in each iteration. The splitting process stops when all the segments are smaller than a certain size τ, which is a pre-defined parameter specifying the maximal number of data points we want the sequential clustering to handle. Empirically we set $\tau = 64$ (see Section 5.2). Fig. 2(d) illustrates the splitting process.

Table II Description and Comparison of Different Versions of D2-Clustering

	D2	**Constrained D2**	**Weighted D2**
Input	N discrete distributions $V_1, V_2, \ldots, V_N$, where $V_i = \{(v_i^{(1)}, p_i^{(1)}), \ldots, (v_i^{(t_i)}, p_i^{(t_i)})\}$		
Output	Cluster labels $L(i) = j, i = 1, \ldots, N, j \in \{1, \ldots, k\}$ and cluster centroids $Z_1, \ldots, Z_k$.		
Step 1	Initialize k centroids.		
Step 2	Allocate a cluster label to each data point V_i: $L(i) = \text{argmin}_j\, D(V_i, Z_j)$.		
Step 3	Solve the LP in Eq. (7) with the constraints in Eq. (8).	Solve the LP in Eq. (7) with the constraints in Eq. (12).	Solve the LP in Eq. (13) with the constraints in Eq. (8).
Step 4	Update supports by Eq. (9).		Update supports by Eq. (14).
Step 5	Go back to Step 3 until centroids are stable.		
Step 6	Evaluate objective Eq. (4).		Evaluate objective Eq. (13).
Step 7	Go back to Step 2 until the objective function reaches a stable value.		

Within each round of the splitting, the binary partitioning is an approximation of, and computationally less expensive than, an optimal clustering. For discrete distributions measured by the Mallows distance, large LPs should be avoided in this step for the time concern. The method we use is a computationally reduced version of D2-clustering. It is fast because of some constraints imposed on the centroids. We refer to this version as the *constrained D2-clustering*.

The constrained D2-clustering method is described in Table II (together with other variants of D2-clustering). The major difference between the constrained D2-clustering and D2-clustering algorithm is that the weights of support vectors in a centroid Z_j are not updated once it is initialized. Therefore the constraints in Equation (8) for centroids update are simplified to:

$$\begin{aligned}
&\sum_{b=1}^{t_i} w_{a,b}^{(i)} = q_j, \forall (i, \alpha) \text{ s.t. } L(i) = j, 1 \le a \le s_j \ , \\
&\sum_{a=1}^{s_j} w_{a,b}^{(i)} = p_i^{(b)}, \forall (i, b) \text{ s.t. } L(i) = j, 1 \le b \le t_i \ , \\
&w_{a,b}^{(i)} \ge 0, \forall i, a, b \ .
\end{aligned} \tag{12}$$

The large LP can then be divided to smaller ones because q_j's are constant in the optimization and the coefficients $w_{a,b}^{(i)}$'s are no longer coupled. Though not optimally clustered, the binary segmentation generally puts similar data points to same groups. As a result, the eventual binary splitting result allocates nearby data points to a same sub-task, in which the local clusters are similar to those obtained by the sequential algorithm in the same vicinity.

4.3. Merging of Parallel Results

Data segments generated in the "divide" phase are distributed to different processors and clustered locally. The local clustering results are merged by a new pass of clustering. In the multi-pass clustering structure, the input data of any pass following the first one are the local cluster centroids from its previous pass. Each centroid represents a cluster containing a certain number of data and the sizes for different clusters are usually different due to the skewness of the real-world data. If we intend to keep equal contribution from each original data point, the cluster centroids passed to a higher level in the hierarchy should be weighted and the clustering method should take those weights into account. We thus extend D2-clustering to a weighted version and use the weighted D2-clustering algorithm as the plug-in clustering method in the algorithm to abstract each data chunk containing $N' \le \tau$ points by N'/R cluster centroids.

It is apparent that with weighted data, the step of nearest neighbor assignment of centroids to data points is unaffected, while the update of the centroids under a given partition of data is. When data V_i have weights ω_i, a natural extension of the optimization in Equation (6) to update a centroid Z_j is:

$$\min_{z_j^{(a)}, q_j^{(a)}, w_{a,b}^{(i)}} \sum_{i:L(i)=j} \omega_i D^2(Z_j, V_i) \ . \tag{13}$$

To solve the optimization, we first fix the support vectors $z_j^{(a)}$'s and update their weights $q_j^{(a))}$'s by solving the linear program with objective (13) under the same constraints specified in Equation (8). Then we update the support vectors $z_j^{(a)}$ in a way that considers the weight ω_i for each data point:

$$z_j^{(a)} = \frac{\sum_{i:L(i)=j} \omega_i \sum_{b=1}^{t_i} w_{a,b}^{(i)} v_i^{(b)}}{\sum_{i:L(i)=j} \omega_i \sum_{b=1}^{t_i} w_{a,b}^{(i)}} \ . \tag{14}$$

Because the optimal solution of a centroid is not affected when the weights are scaled by a common factor, we simply use the number of original data points assigned to each centroid as its weight. The weights can be computed recursively when the algorithm goes through levels in the hierarchy. At the bottom level, each original data point is assigned with the weight 1. The weight of a data point in a parent level is the sum of weights of its children data points.

Table II describes and compares the workflow of the D2-clustering, constrained D2-clustering, and weighted D2-clustering in detail side by side. Both latter algorithms, as variants of the original D2-clustering algorithm, work in the same way as D2-clustering. Different versions of D2-clustering vary in the definitions of objective functions, and the feasible solution spaces.

4.4. Stopping Criteria

After each pass of the parallel "divide" and "merge" operations, a set of cluster centroids are acquired. They are treated as the new set of data points if the clustering proceeds to the next pass. A root node will evaluate the quality of the current clustering result and determine whether to continue or stop the clustering. If certain stopping criteria are met, the program will terminate and output the results. Otherwise another pass of parallel clustering is invoked to further merge the current set of centroids.

Three clustering properties are checked as the stopping criteria.

(a) Minimal number of clusters k_{min}. The clustering stops if cluster number $k_l \leq k_{min}$.
(b) Maximal cluster mass σ. Define the mass of a cluster as the number of original input data it contains. If any cluster's mass is larger than σ, the clustering stops.
(c) Maximal within-cluster dispersion δ. If the mean squared distance from original the data points to the centroid of any cluster is large than δ, the clustering stops.

The first criterion is similar to setting an pre-determined cluster number. The other two criteria are both simple cluster quality evaluations. The clustering process stops if any of the three criteria is met. The cluster qualities (b) and (c), i.e., the cluster mass and dispersion, can be recursively estimated (see Appendix B). Therefore it costs the root node little extra computation to check these criteria.

The parameters k_{min}, σ, and δ are set by the prior knowledge of the dataset and the expected clustering result. The clustering hierarchy can be terminated at any level by setting different stopping criteria for different needs. Nevertheless, the clustering run-time is dominated by the first pass of clustering because the hierarchy shrinks

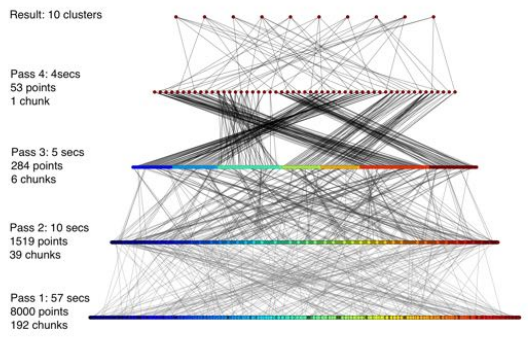


Fig. 3 . Visualization of the clustering hierarchy. Each row of dots represent the discrete distributions being clustered at a certain pass. Points with the same color are assigned to a same chunk for local clustering. Lines between adjacent rows indicate the cluster assignment of a single pass clustering. Data points connected to a same point are grouped in the same cluster and represented by the higher-level point in the following pass of clustering.

exponentially. In each pass, data are divided to chunks with similar sizes and clustered locally. The run-time therefore contains a term corresponding to the dividing operation and a linear term corresponding to the parallel tasks, making the overall run-time close to linear. Detailed run-time analysis is given in Appendix A.

5. EXPERIMENTS

We conducted several experiments to quantitatively evaluate the PD2-clustering algorithm and demonstrate its advantages in real-world applications. We use synthetic datasets to benchmark different clustering algorithms by data with ground-truth labels. The random generation of discrete distributions and the benchmarking metrics are introduced in Appendix D. We then demonstrate the usage and advantages of PD2-clustering in two real-world large-scale multimedia applications, concept modeling of Web images, and protein sequence clustering.

All experiments run on a computing cluster with in total 2,048 CPU cores of 2.7 GHz clock frequency[4]. Based on the server's configuration, the PD2-clustering algorithm is implemented by MPI and the embedded LP is optimized by the IBM CPLEX optimizer.

5.1. Visualization of Clustering Workflow

Before quantitatively evaluating the algorithm, we first visualize the workflow of a real clustering process in order to provide readers with an intuitive interpretation of the PD2-clustering algorithm. We conducted the clustering on the synthetic dataset with 8,000 discrete distributions, and recorded the results for each pass of parallel clustering. The parallel clustering is deployed on 16 CPU cores. We set local chunk size $\tau = 64$, and shrinking factor $R = 5.0$. Fig. 3 shows the visualization. Data chunks are distinguished by different colors, and the cluster assignment in each pass is represented by the lines connecting each child data point to its corresponding local centroid, which is further clustered in the following pass. The hierarchy shrinks pass by pass, so does the run-time. Eventually several global cluster centroids are obtained when the number of data points is small enough at a certain pass. As analyzed in Appendix A, the overall run-time is dominated by the run-time of the first pass.

[4]In practice the system limits the maximal number of processors each user can occupy at a time to 32.

5.2. Impacts of Chunk Size and Shrinking Factor

The size of local parallel chunk τ and the shrinking factor R are two important parameters that affect the run-time, as well as the clustering quality, of the PD2-clustering algorithm. Intuitively, a large chunk size will introduce less approximation, and when $\tau = N$, PD2-clustering is identical to sequential D2-clustering. However, the overall clustering run-time will also increase as τ increases. The shrinking factor R controls how fast the clustering hierarchy shrinks and how many data points each centroid summarizes in each pass. A large R reduces the number of passes, making the clustering terminates faster, but introduces more approximation within each pass. A small R increases the number of passes and introduces less abstraction in each pass.

On a synthetic dataset containing 128,000 discrete distributions in 10 clusters, we tested PD2-clustering with different combinations of chunk sizes and shrinking factors. We use the Adjusted Rand Index (ARI, higher is better) and the point-to-centroid Average Squared Distance (ASD, lower is better) to evaluate the clustering quality with respect to the ground-truth. The dataset and evaluation metrics are explained in Appendix D. The results are plotted in Fig. 4, from which we can conclude that except the $\tau = 16$ case where the chunk size is too small and makes the performance random, generally the combination of a medium R and large τ achieves a good clustering quality, which validates our intuition. Besides the clustering quality, run-time is another factor to be taken account when choosing parameters R and τ. For such sake we need to limit τ within a medium range to prevent local clustering tasks being too slow. In the following experiments, we set $R = 5.0$ and $\tau = 64$ because such a combination has a balanced clustering performance in terms of both quality and run-time[5].

Fig. 5 shows the ARI curves (with respect to R) of PD2-clustering on datasets with different distributions. For a certain chunk size, the ARI curves on different datasets are all of similar shapes. As a result, same set of parameters have similar impacts on different distributions and we empirically apply $R = 5.0$ and $\tau = 64$ to any dataset. Detailed analysis of R's and τ's impacts are presented in Appendix A and C.

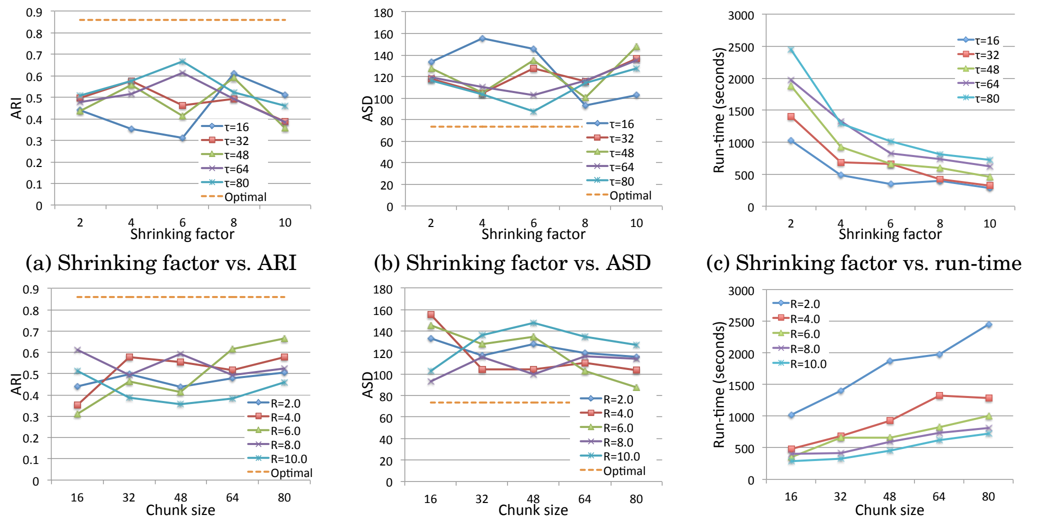
5.3. Scalability

To test the scalability properties of the PD2-clustering algorithm, we ran the clustering on synthetic datasets with sizes of 500, 1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000, 256000, 512000, and 1024000. We also tested several other clustering algorithms on these datasets and compared them with PD2-clustering. The following algorithms are included and compared in the experiment:

(*A0*) The PD2-clustering algorithm;
(*A1*) The sequential D2-clustering algorithm;
(*A2*) The constrained D2-clustering algorithm introduced in in Section 4.2;
(*A3*) PD2-clustering with random segmentation;
(*A4*) The PKMeans clustering algorithm [Zhao et al. 2009] after applying vector quantization (VQ) [Gersho and Gray 1992] to the data;
(*A5*) The parallel spectral clustering algorithm [Chen et al. 2011].

Except the original D2-clustering algorithm which is sequentially performed by a single CPU, all the other algorithms are parallel algorithms. We use 16 CPU cores to run each of the parallel algorithms for a fair run-time comparison, and the results are plotted in Fig. 6(c). The sequential D2-clustering algorithm, unsurprisingly, is the

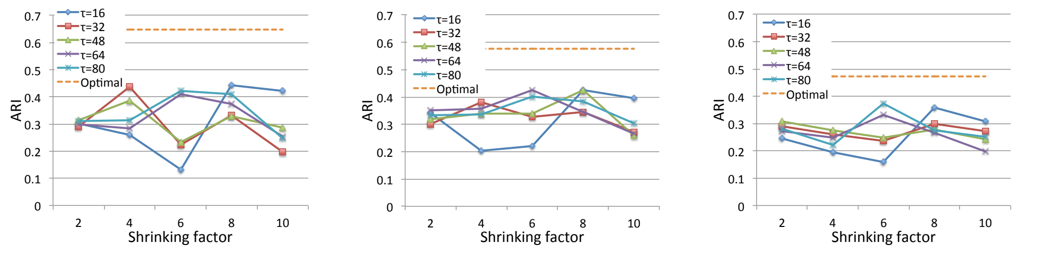
[5]In Fig. 4 $R = 6.0$ and $\tau = 80$ achieves best clustering quality. For $\tau = 64$ the clustering quality is only slightly worse, but the algorithm runs much faster. When the size of dataset N is in the order of 10^4, the numbers of passes ($\lceil \log_R \frac{N}{k} \rceil$, $k \simeq 10$) are likely to be the same for both $R = 5.0$ and $R = 6.0$. We choose $R = 5.0$ for less approximation within a single pass.



(a) Shrinking factor vs. ARI (b) Shrinking factor vs. ASD (c) Shrinking factor vs. run-time

(d) Chunk size vs. ARI (e) Chunk size vs. ASD (f) Chunk size vs. run-time

Fig. 4 . The impacts of chunk size (τ) and shrinking factor (R) on the performance of PD2-clustering. The synthetic dataset contains 128,000 data points in 10 clusters. 7% data from different clusters are overlapped. Each point in the plots are the average of 5 runs for the corresponding parameters. (a-c) The impact of R on Adjusted Rand Index (ARI), Average Squared Distance (ASD), and run-time. (d-f) The impact of τ on ARI, ASD, and run-time. Higher ARI and lower ASD mean better clustering quality. The optimal ARI and ASD are plotted in dashed lines. Medium R and τ are preferred considering both cluster quality and run-time.

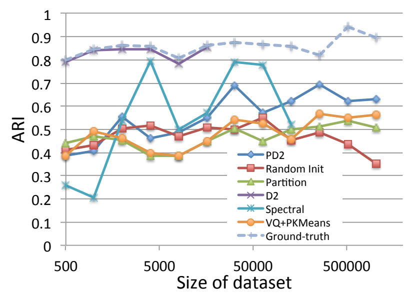


(a) 19% overlap (b) 25% overlap (c) 33% overlap

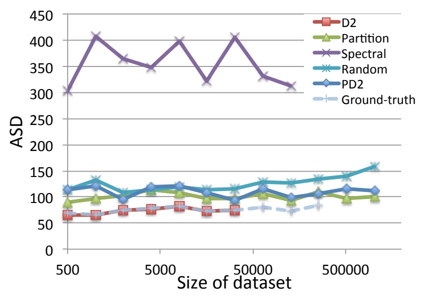
Fig. 5 . The clustering ARI (average of 5 runs) on synthetic datasets with different distributions. All sets contain 128,000 data points and 10 clusters. They are generated with different variances, and have different amounts of overlapping data. Though the optimal ARIs (in dash lines) on these datasets are different, the impacts of chunk size (τ) and shrinking factor (R) are similar. The shapes (trends) of the curves in Fig. 5 (a-c) and Fig. 4 (a) resemble each other.

slowest one among the tested algorithms due to the expensive centroid update operation. In the experiment it fails to complete a clustering on the dataset with 64,000 data points within a time limit of 6 hours. The spectral clustering is also slow even though it is implemented in parallel, because all the pairwise Mallows distances between data points and a large eigen-decomposition need to be computed. It can barely complete the clustering of 128,000 data points in 6 hours and cannot efficiently handle larger datasets. The rest four parallel algorithms can all cluster datasets containing as many as 1,024,000 data points within a short time. Though the PD2-clustering algorithm is more expensive than the constrained D2-clustering algorithm and the parallel K-means algorithm, they are of a similar order in time complexity.

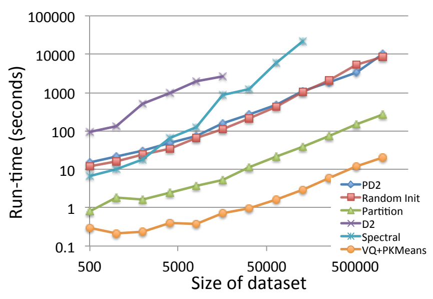
As to the clustering quality, the sequential D2-clustering algorithm achieves the best ARI and ASD which is closest to the optimal values (when the dataset is small enough



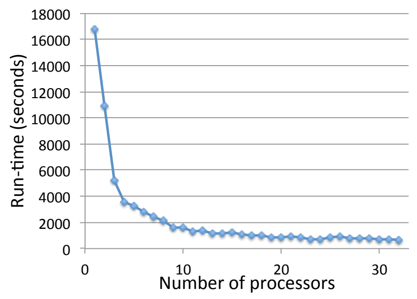
(a) ARI vs. size of dataset



(b) ASD vs. size of dataset



(c) Run-time vs. size of dataset



(d) Run-time vs. CPU number

Fig. 6 . Scalability analysis of PD2-clustering. PD2-clustering is tested on different scales of data and compared with other clustering approaches, including sequential D2-clustering (D2), constrained D2-clustering used as the partitioning method in PD2-clustering (Partition), PD2-clustering with random segmentation (Random Init). parallel K-means [Zhao et al. 2009] after quantizing the discrete distributions to vectors (VQ+PKMeans, the vector quantization time is not counted), and parallel spectral clustering [Chen et al. 2011]. For each case the result is the average of 5 runs. 16 parallel CPU cores are employed for all parallel algorithms. Sequential D2-clustering and parallel spectral clustering cannot finish within a reasonable time when the scale is large. (a) Comparison of clustering ARI. The results from sequential D2-clustering gains the best clustering quality. PD2-clustering outperforms constrained D2-clustering and VQ+PKmeans. (b) Comparison of clustering ASD (among methods using Mallow distance). D2-clustering obtains nearly optimal cluster tightness and PD2-clustering's results are relatively tight. (c) Run-time comparison. Both axes are in logarithmic scales. PD2-clustering's time complexity has a much lower order than D2-clustering's. (d) PD2-clustering's run-time on 128,000 data points with different numbers of CPU cores employed. The number of processors is inversely proportional to the overall run-time.

to be clustered). All other algorithms achieve less optimal clusters by trading off some accuracy for speed. The spectral clustering algorithm, which is based on the pair-wise distances among data points, does not have a consistent performance on clustering ARI (presumably because the parallel implementation only utilizes the distances between neighboring points for fast computation) and the clusters are very loose[6]. Similarly, the VQ step before PKMeans in (*A4*) discards some information from the discrete distributions when transforming them to vectors. The ARI consequently deteriorates. The curves for approach (*A2*) and (*A3*) demonstrate the necessity of adopting constrained D2-clustering to divide data. None of the approach (*A2*) and (*A3*) standalone achieves the ARI of PD2-clustering which combines them. Comparing (*A0*), (*A2*) and (*A3*)'s ASD curves in Fig. 6(b), we can see that partitioning data by constrained D2-clustering conserves the cluster tightness, which is crucial to make PD2-clustering's results tight.

Fig. 6(d) plots the run-time of PD2-clustering on the dataset containing 128,000 data points with different numbers of parallel CPU cores. When the number of CPU cores is

[6]Cluster centroids for computing ASD of spectral clustering are the data points whose projections in the spectral eigenspace are closest to the K-means' centroids in the space.

smaller than the number of parallel sub-tasks (i.e., the number of data chunks), each CPU needs to sequentially handle several sub-tasks, which consequently makes the overall run-time longer. Particularly when 1 CPU is employed, we only take advantage of the divide-and-conquer structure but no computation is done in parallel. Still the parallel algorithm can complete the clustering of 128,000 data points, which the sequential algorithm cannot handle. The curve in Fig. 6(d) shows an inverse proportional relationship between the run-time and the CPU number. Although in our experiment the maximal number of CPUs we can request is limited to 32, theoretically the scalability of PD2-clustering can be always improved if more CPUs are involved.

5.4. Application 1: Large-scale Image Clustering and Concept Learning

The PD2-clustering algorithm is useful in applications where the data points are represented by discrete distributions (or BoWs) and measured by the Mallows distance. In this subsection we demonstrate its usage on image clustering. The experiment is conducted on the NUS-WIDE dataset [Chua et al. 2009], in which there are in total 269,648 images from Flickr and each image is manually annotated with one or more ground-truth tags from a list of 81 common concepts. 161,789 images in the dataset are used as training data and the rest are used for testing.

Each subset of the training data containing images with the same tags are clustered by PD2-clustering. Each image descriptor I_i is composed by three bags of weighted vectors corresponding to the image's color (in the LUV color space), texture (Daubechies-4 wavelet [Daubechies 1992] coefficients), and key points (SIFT descriptors [Lowe 2004]) components[7]. Denote the discrete distributions for color, texture, and key points as $U_i = \{(u_i^{(1)}, p_{u,i}^{(1)}), \ldots, (u_i^{(t_v)}, p_{u,i}^{(t_u)})\}$, $V_i = \{(v_i^{(1)}, p_{v,i}^{(1)}), \ldots, (v_i^{(t_v)}, p_{v,i}^{(t_v)})\}$, and $Y_i = \{(y_i^{(1)}, p_{y,i}^{(1)}), \ldots, (y_i^{(t_y)}, p_{y,i}^{(t_y)})\}$. I_i is the concatenation of these three modalities, i.e., $I_i = \{U_i, V_i, Y_i\}$[8]. The distance between I_i and I_j is defined as:

$$\hat{D}(I_i, I_j) = (D^2(U_i, U_j) + D^2(V_i, V_j) + D^2(Y_i, Y_j))^{\frac{1}{2}} , \tag{15}$$

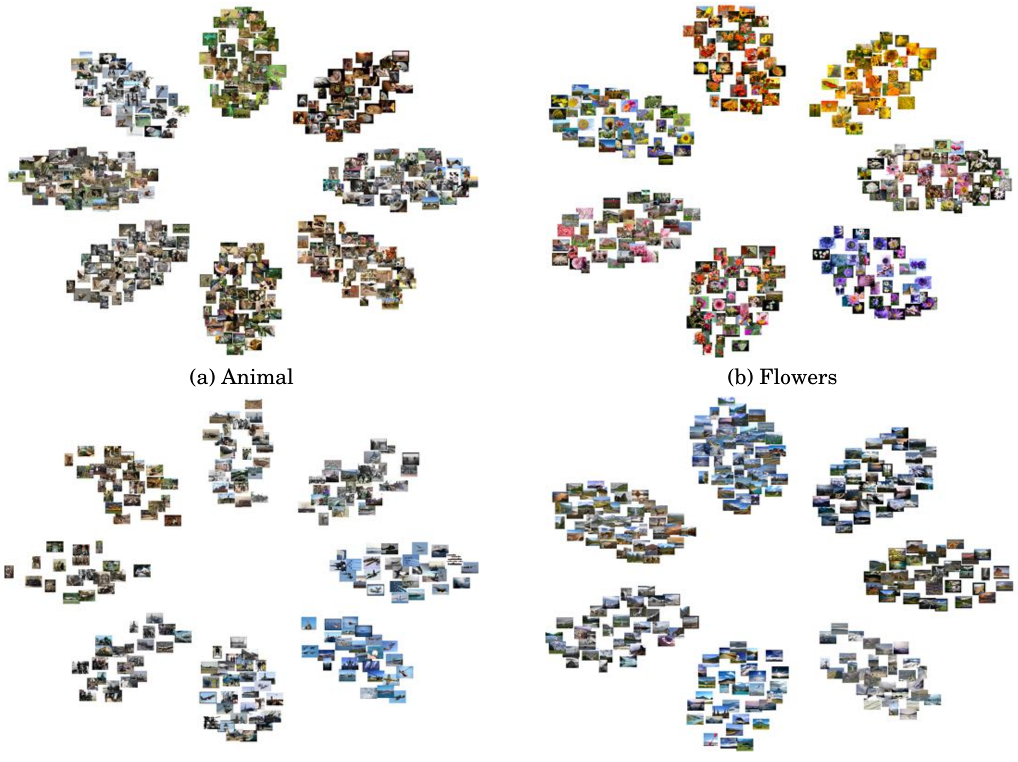
where $D(\cdot,\cdot)$ is the Mallows distance. The PD2-clustering algorithm can be extended to the case of multiple discrete distributions by using $\hat{D}(\cdot,\cdot)$ as the metric and performing the centroid update operation (as in Equation (7)) respectively only within each modality. See [Li and Wang 2008] for the details.

In the training set, the average size of image subset being clustered is 3,175, much larger than the scale which the original D2-clustering can handle. It takes on average 200 seconds for PD2-clustering to complete. After that each subset of images are grouped into clusters. Because there is no ground-truth cluster labels for each subset, we evaluate the clustering result in three ways.

First, we visualize the clustering results by drawing representative images on a 2D space. For each image subset, we arrange representative images of each cluster generated by PD2-clustering in a 2D space and put images in the same cluster together. Visualizations of several subsets are shown in Fig. 7. The visualizations qualitatively show that images are grouped correctly by their visual traits.

Second, we quantify the clustering quality by the cluster tightness, i.e., the average squared distance (ASD) between the data points and the corresponding centroids. We also apply sequential D2-clustering and parallel spectral clustering on the training set and compare the ASDs obtained by different algorithm. Because the sequential D2-clustering algorithm is slow, it is only applied on 200 random images in each concept.

[7]We perform K-means on all pixel-level features (for color and texture) or key point DOG descriptors (for SIFT), then use the cluster centroids and sizes as the supports and weights for the discrete distributions.
[8]The support vectors for each modality are normalized so that they have equal contribution.

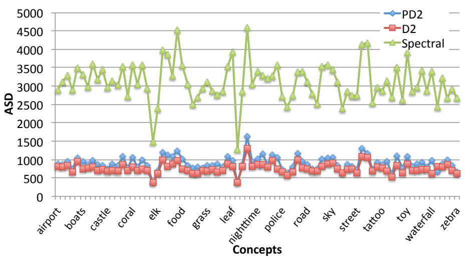

(a) Animal (b) Flowers

(c) Military (d) Mountain

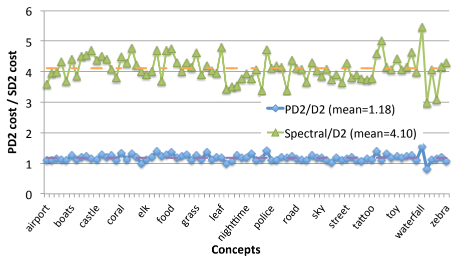
Fig. 7 . Visualization of the image clustering results. Large clusters for four concepts are visualized. Image clusters are arranged in a circular layout. Within each cluster, images not significantly overlapped are draw around the cluster center based on their visual distances to the centroid.

The sequential D2-clustering roughly takes the same amount of time as that used by PD2-clustering on the whole dataset. For parallel spectral clustering, we use the data entries whose projections in the spectral eigenspace are closest to the cluster centers in the space as the centroids for evaluating ASD. Fig. 8(a) shows the ASD evaluation results. The sequential D2-clustering algorithm obtains the tightest clusters. The PD2-clustering results are only slightly less tight than the sequential D2-clustering's results. Parallel spectral clustering does not aim at optimizing the tightness of clusters, therefore obtains the least tight clusters. The relative tightness of PD2-clustering and parallel spectral clustering's results compared with the D2-clustering's results is plotted in Fig. 8(b). Considering the sequential algorithm's result is close to optimal (shown in Fig. 6), PD2-clustering is much more optimal than parallel spectral clustering in terms of cluster tightness.

Third, we apply the clustering results to the image annotation application and use the annotation results to demonstrate the effectiveness of PD2-clustering. Given the image clusters of a certain concept, we adopt the Hypothetical Local Mapping (HLM) method introduced in [Li and Wang 2008] to train image concept models. The image clusters for each concept c are used to estimate a Gaussian Mixture Model (GMM), $g_c(I)$, where I is the image descriptor. Given a test image with the descriptor I_i, we compute the posterior probability of I_i belonging to concept c, $P(I_i|\text{concept} = c) = g_c(I_i)$, for each concept c and rank the probabilities. Then the image is annotated by the top ranked concepts. Same as the ASD evaluation, the image annotation result from PD2-clustering is also compared with the results from sequential D2-clustering



(a) Absolute ASD



(b) Relative ASD

Fig. 8 . Tightness of image clustering results. PD2-clustering, D2-clustering, and parallel spectral clustering are performed on 81 image subsets. The average squared Mallows distance (ASD) from images to cluster centroids are computed and plotted for each algorithm on each image subset. (a) The absolute ASD. PD2-clustering and D2-clustering results are both tight. The clusters obtained by spectral clustering are much looser than the other two sets of results. (b) The ASD ratio between PD2-clustering and D2-clustering, and that between spectral clustering and D2-clustering. On average the PD2-clustering's ASD is 1.18 times of D2-clustering's ASD, compared with the 4.10 times difference between spectral clustering and D2-clustering.

Table III Image Annotation Performance by Different Clustering Approaches

	PD2	**D2 (small set)**	**K-means (+VQ)**	**Spectral**
Run-time (seconds)	16178	10414	682	82800
Top 5 tagging precision	**10.2%**	5.0%	7.4%	6.1%
Correctly tagged images	**40.2%**	22.8%	32.7%	27.2%

Note: The training dataset contains 161,789 Flickr images with 81 manually labeled concepts and the testing dataset contains 107,859 images. PD2-clustering, K-means, and spectral clustering are all implemented in parallel and run on 16 CPU cores. D2-clustering runs on a single CPU core and only processes 200 random images per concept due to its high time complexity. K-means is applied on the quantized data vectors. For spectral clustering, image descriptors whose projections are closest to the cluster centers in the spectral eigenspace are used as cluster centroids for modeling. Top 5 tags for each image are compared with its ground-truth tags and the precision is listed on the second row. An image is counted as a correctly tagged one if at least one predicted tags matches the ground-truth.

(on 200 random images of each concept) and parallel spectral clustering. Additionally, we quantize the image descriptors to vectors, apply K-means to cluster the data, and use the Euclidean distance from each quantized data entry to its closest centroid in the vector space for concept modeling. The annotation performance and sample results are shown in Table. III and Fig. 9. The PD2-clustering algorithm, although introduces some approximation for parallelization, learns better concept models than the sequential D2-clustering algorithm because it efficiently leverages more training data. Compared with K-means and spectral clustering, PD2-clustering algorithm better reveals the distribution of BoW descriptors, therefore obtains better annotation performance.

5.5. Application 2: Protein Sequence Clustering

In bioinformatics, composition-based methods for sequence clustering and classification (either protein [Garrow et al. 2005] or DNA [Kelley and Salzberg 2010]) use the frequencies of different compositions in each sequence as the signature. The frequency histogram of nucleotides or amino acids, which are the basic components of DNA or protein, are adopted as the descriptor of a DNA or protein sequence in these methods. Because different nucleotide or amino acid pairs are related due to their molecular structures and evolutionary relationships, cross-term relations should be considered when we compute the distance between two such histograms. As a result, the Mallows distance would be a proper metric in composition-based approaches.

For protein sequence clustering, because the vocabulary size of protein words (each contains several amino acids) is large and the composition-based representation is sparse, composition-based approaches are not widely applied. Most existing approaches rely on the pair-wise sequence alignment to perform connectivity-based

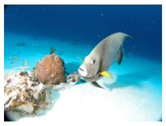
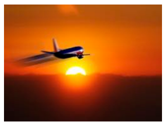
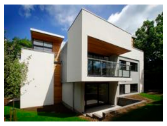
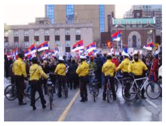
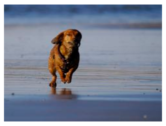
					
PD2	**animal** surf **coral**	**sunset** **sun** beach	**house** airport **tree**	**protest** **street** cars	**running** **reflection** plane
D2	sunset sky road	street elk beach	train horses **town**	fish grass rocks	rocks **water** rainbow
K-means	**water** **animal** moon	moon birds fish	protest book water	**protest** **person** animal	**water** earthquake fire
Spectral	sunset toy castle	sand fox **sun**	castle computer ocean	castle computer ocean	plane castle flags

Fig. 9 . Sample image tagging results. Concept models learned by PD2-clustering, D2-clustering, K-Means, and spectral clustering are used to tag the same set of testing images. Top 3 tags predicted by each model are listed. Relevant tags are marked in bold. Models obtained from PD2-clustering achieve best performance. Images courtesy of Flickr users Pieter de Boer (unknowndiver), Ben Cutshall (ben cutshall photos), Darrell Godliman, Scott Snider (sniderscion), and Saskia Scholte (Buikschuivers).

clustering (e.g., hierarchical clustering [Enright and Ouzounis 2000] and spectral clustering [Paccanaro et al. 2006]). These approaches only work on small datasets (which typically contain about 1,000 protein sequences) because computing the alignment for all pairs of sequences is expensive. CD-HIT [Huang et al. 2010], as a state-of-the-art protein sequence clustering algorithm that can run on large datasets, though uses the short protein word comparison as a first stage clustering to improve the scalability, still relies on sequence alignment to further cluster potentially matched pairs selected by the rough word comparison. These alignment-based algorithms usually fail to work when we hope to summarize a large dataset by a small number of clusters because sequence alignment is only capable of discovering closely matched proteins. In such cases, PD2-clustering is appealing as it can efficiently cluster sparse composition-based protein descriptors. In this experiment, we performed PD2-clustering on protein sequences from the overlapping set of the Swiss-Prot database [Boeckmann et al. 2003] and the Prosite database [Sigrist et al. 2013]. The Swiss-Prot database contains 519,348 proteins, many of which are categorized into different protein families by the Prosite database. In total the overlapping set of these two databases contains 204,035 labeled proteins from 1,296 classes. We selected 10,742 sequences from 3 largest classes (with 4031, 3608, and 3103 sequences respectively) for this experiment.

For each protein sequence, we count the occurrence of each type of amino acid, dipeptide (tuple of amino acids), and tripeptide (triplet of amino acids), and use the frequency histograms of these basic sequence components, $\{V^{(1)}, V^{(2)}, V^{(3)}\}$, as the descriptors ($V^{(1)}$, $V^{(2)}$ and $V^{(3)}$ correspond to the histograms for amino acids, dipeptides, and tripeptides respectively, where the superscript for each histogram indicates the word length of the supports). Because there are 20 types of amino acids, $V^{(1)}$ has a fixed length of 20. For dipeptides and tripeptides, the vocabulary sizes are 20^2 and 20^3. To make the representation compact, we only adopt the most frequent 32 (dipeptide or tripeptide) words to build histograms $V^{(2)}$ and $V^{(3)}$. The distance between two sequence descriptors $\boldsymbol{V_i} = \{V_i^{(1)}, V_i^{(2)}, V_i^{(3)}\}$ and $\boldsymbol{V_j} = \{V_j^{(1)}, V_j^{(2)}, V_j^{(3)}\}$ is computed as:

$$\tilde{D}(\boldsymbol{V_i}, \boldsymbol{V_j}) = T(V_i^{(1)}, V_j^{(1)}) + T(V_i^{(2)}, V_j^{(2)}) + T(V_i^{(3)}, V_j^{(3)}) \;, \tag{16}$$

where $T(\cdot, \cdot)$ is the transportation distance (defined in Equation (2)) between two histograms of the same type. The cost between two supports is computed by their sequence matching score. Denote the a-th support for histogram $V_i^{(X)}$ (a word containing

Table IV Performance of Different Protein Sequence Clustering Approaches

	PD2	**Spectral**	**CD-HIT**	**D2 (900 samples)**
Run-time (seconds)	1048	273656	295	1415
Number of clusters	10	10	654	10
Largest cluster size	4399	1259	824	–
Cluster ARI	0.202	0.051	0.097	0.176
ASD (Mallows)	8.369	–	–	7.750

Note: The clustering is performed on 10,742 protein sequences from 3 Prosite classes, with 16 CPU cores employed in the computation. The Mallows distance adopted by PD2-clustering is defined in Equation (16). The pair-wise protein distance used in spectral clustering is computed by BLAST. Sequential D2-clustering is applied on a subset with 900 randomly selected sequences.

X amino acid symbols, $X = 1, 2, 3$) as $\vec{v}_{i,a}^{(X)} = v_{i,a}^{(1)} \dots v_{i,a}^{(X)}$, where each $v_{i,a}^{(x)}$ is a certain amino acid. The similarity between two supports $\vec{v}_{i,a}^{(X)}$ and $\vec{v}_{j,b}^{(X)}$ is defined as:

$$S(\vec{v}_{i,a}^{(X)}, \vec{v}_{j,b}^{(X)}) = \frac{1}{X} \sum_{x=1}^{X} s(v_{i,a}^{(x)}, v_{j,b}^{(x)}) \ , \tag{17}$$

where the evolutionary similarity between the two amino acids, $s(v_{i,a}^{(x)}, v_{j,b}^{(x)})$, is given in the PAM250 scoring matrix [Pearson 1990]. The distance between these two supports is then defined as:

$$C(\vec{v}_{i,a}^{(X)}, \vec{v}_{j,b}^{(X)}) = \frac{1}{2}\left(S(\vec{v}_{i,a}^{(X)}, \vec{v}_{i,a}^{(X)}) + S(\vec{v}_{j,b}^{(X)}, \vec{v}_{j,b}^{(X)})\right) - S(\vec{v}_{i,a}^{(X)}, \vec{v}_{j,b}^{(X)}) \ . \tag{18}$$

The cost function (18) is adopted in the calculation of the transportation distance (16). Additionally, PD2-clustering for the composition-based protein descriptors are performed based on such metric. When updating centroids in the clustering loop, the weights and locations of the supports for a centroid are iteratively updated as specified in Section 2.3. Within each centroid update iteration, the supports' weights for a centroid are first updated by the linear program introduced in Equation (8). Being symbolic, the supports themselves are then updated by searching the word in the dictionary that can minimizes the objective in Equation (7) with updated weights. This results in a minor modification of Step 4 in the algorithm described in Table II.

We tested two other methods on the same dataset. First, we followed the approach introduced by Paccanaro et al. [2006] to invoke the BLAST library for aligning all pairs of protein sequences, and use the E-value returned by it as the distance between two sequences. With all the pair-wise distances, we use parallel spectral clustering to cluster the data. Both the computation of pair-wise distances and the spectral clustering are parallelized by 16 CPU cores. Secondly, we applied the CD-HIT program [Huang et al. 2010] to do the clustering. As mentioned above, CD-HIT groups sequences by their alignment similarities. A similarity threshold is needed as the input. Only sequences whose similarity scores are above the threshold are grouped into same clusters. In order to achieve larger clusters, we set the threshold to the lowest possible value (0.4). We expect the clusters agree with Prosite classes to some extent, though perfectly matched clusters to the manually labeled classes are difficult to achieve.

Table IV lists the clustering results of these three approaches. With 16 CPU cores employed, it costs a clock time of 1,048 seconds for PD2-clustering to discover 10 clusters. The run-time is much shorter than the parallel spectral clustering, and comparable to CD-HIT. In terms of the clustering quality, PD2-clustering discovers 10 clusters with relatively similar sizes. The ARI score is 0.202 compared with the labels from the Prosite database. In contrast, the spectral clustering approach is extremely slow and incompetent for large-scale applications. The CD-HIT approach, though running fast, cannot further group the dataset after 654 clusters are formed. In fact 284 out of the 654 clusters contain only one sequence, and the largest cluster size is 824. A low ARI score is obtained due to the fragmented clusters. This indicates the inefficiency of CD-HIT in clustering and abstracting large sparse datasets. The ASD comparison[9] between PD2-clustering and D2-clustering (on a subset), as we did in the image clustering experiment in Section 5.4, shows that PD2-clustering achieves tight clusters. The results shows PD2-clustering's advantages on both the run-time and the clustering quality in protein sequence clustering over the state-of-the-art approaches.

6. CONCLUSION

A novel parallel D2-clustering algorithm with dynamic hierarchical multi-pass structure is proposed. The algorithm can efficiently perform clustering operations on data represented as discrete distributions. Three research questions presented in Section 1 are answered by the analysis and experiments in the paper. First (RQ1), the multi-pass divide-and-conquer algorithm structure and the introduction of parallel computing significantly improve the scalability of D2-clustering. The parallel algorithm is orders of magnitude faster than the original sequential D2-clustering algorithm. By deploying the parallel algorithm on multiple CPU cores, the time complexity of D2-clustering can be improved from high-ordered polynomial to close-to-linear time. Second (RQ2), because sub-tasks are divided based on the data points' adjacency in each pass, the parallelization introduces little approximation. Finally (RQ3), we show the potential of PD2-clustering in applications where the Bag-of-Words model is adopted, such as image concept learning and protein sequence clustering. In addition to partitioning data to clusters, PD2-clustering also generates representatives for each cluster, which is useful for statistical modeling and data abstraction. The scalability advantage of PD2-clustering makes it competent to discover knowledge from big data.

REFERENCES

D. Arthur, B. Manthey, and H. Röglin. 2009. K-means has polynomial smoothed complexity. In *50th Annual IEEE Symposium on Foundations of Computer Science*. 405–414.

A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh. 2005. Clustering with Bregman divergences. *The Journal of Machine Learning Research* 6 (2005), 1705–1749.

T. Batu, L. Fortnow, R. Rubinfeld, W. D. Smith, and P. White. 2013. Testing closeness of discrete distributions. *J. ACM* 60, 1 (2013), 4:1–4:25.

C. Beecks, A. M. Ivanescu, S. Kirchhoff, and T. Seidl. 2011. Modeling multimedia contents through probabilistic feature signatures. In *Proceedings of the 19th ACM International Conference on Multimedia*. 1433–1436.

B. Boeckmann, A. Bairoch, R. Apweiler, M. C. Blatter, A. Estreicher, E. Gasteiger, M. J. Martin, K. Michoud, C. O'Donovan, I. Phan, S. Pilbout, and M. Schneider. 2003. The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucleic Acids Research* 31, 1 (2003), 365–370.

W. Chen, Y. Song, H. Bai, C. Lin, and E. Y. Chang. 2011. Parallel spectral clustering in distributed systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 3 (2011), 568–586.

T. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng. 2009. NUS-WIDE: a real-world web image database from National University of Singapore. In *Proceedings of the ACM International Conference on Image and Video Retrieval*. 48.

P. Clement and W. Desch. 2008. An elementary proof of the triangle inequality for the Wasserstein metric. *Proc. Amer. Math. Soc.* 136, 1 (2008), 333–339.

T. M. Cover and J. A. Thomas. 2012. *Elements of Information Theory*. John Wiley & Sons.

E. Dahlhaus. 2000. Parallel algorithms for hierarchical clustering and applications to split decomposition and parity graph recognition. *Journal of Algorithms* 36, 2 (2000), 205–240.

I. Daubechies. 1992. *Ten Lectures on Wavelets*. SIAM.

I. S. Dhillon and D. S. Modha. 2001. Concept decompositions for large sparse text data using clustering. *Machine learning* 42, 1-2 (2001), 143–175.

[9]Spectral clustering and CD-HIT adopt other metrics than Mallows distance. Therefore only the ASDs of PD2-clustering and D2-clustering which both adopt Mallows distance are compared.

A. J. Enright and C. A. Ouzounis. 2000. GeneRAGE: a robust algorithm for sequence clustering and domain detection. *Bioinformatics* 16, 5 (2000), 451–457.

R. L. Ferreira Cordeiro, C. Traina Junior, A. J. Machado Traina, , J. López, U. Kang, and C. Faloutsos. 2011. Clustering very large multi-dimensional datasets with mapreduce. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 690–698.

A. Garrow, A. Agnew, and D. Westhead. 2005. TMB-Hunt: An amino acid composition based method to screen proteomes for Beta-Barrel transmembrane proteins. *BMC Bioinformatics* 6, 1 (2005), 56–71.

A. Gersho and R. M. Gray. 1992. *Vector Quantization and Signal Compression*. Springer.

E. Gonina, G. Friedland, E. Battenberg, P. Koanantakool, M. Driscoll, E. Georganas, and K. Keutzer. 2014. Scalable multimedia content analysis on parallel platforms using python. *ACM Transactions on Multimedia Computing, Communications, and Applications* 10, 2 (2014), 18.

S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. 2003. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering* 15, 3 (2003), 515–528.

Y. Huang, B. Niu, Y. Gao, L. Fu, and W. Li. 2010. CD-HIT Suite: A web server for clustering and comparing biological sequences. *Bioinformatics* 26, 5 (2010), 680–682.

L. Hubert and P. Arabie. 1985. Comparing partitions. *Journal of Classification* 2, 1 (1985), 193–218.

L. V. Kantorovich. 1942. On the transfer of masses. In *Dokl. Akad. Nauk. SSSR*. 227–229.

D. Kelley and S. Salzberg. 2010. Clustering metagenomic sequences with interpolated Markov models. *BMC Bioinformatics* 11, 1 (2010), 544–555.

E. Levina and P. Bickel. 2001. The earth mover's distance is the Mallows distance: some insights from statistics. In *Proceedings of 8th IEEE International Conference on Computer Vision*, Vol. 2. 251–256.

J. Li and J. Z. Wang. 2008. Real-time computerized annotation of pictures. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30, 6 (2008), 985–1002.

Y. Linde, A. Buzo, and R. Gray. 1980. An algorithm for vector quantizer design. *IEEE Transactions on Communications* 28, 1 (1980), 84–95.

D. G. Lowe. 2004. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision* 60, 2 (2004), 91–110.

J. MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, Vol. 1. 281–297.

G. Monge. 1781. *Mémoire sur la théorie des déblais et des remblais*. De l'Imprimerie Royale.

K. G. Murty. 1983. *Linear Programming*. Vol. 57. Wiley New York.

C. F. Olson. 1995. Parallel algorithms for hierarchical clustering. *Parallel Comput.* 21, 8 (1995), 1313–1325.

A. Paccanaro, J. A. Casbon, and M. A. Saqi. 2006. Spectral clustering of protein sequences. *Nucleic Acids Research* 34, 5 (2006), 1571–1580.

W. R. Pearson. 1990. Rapid and sensitive sequence comparison with FASTP and FASTA. *Methods in Enzymology* 183 (1990), 63–98.

S. L. K. Pond, K. Scheffler, M. B. Gravenor, A. F. Y. Poon, and S. D. W. Frost. 2010. Evolutionary fingerprinting of genes. *Molecular Biology and Evolution* 27, 3 (2010), 520–536.

A. Rosenberg and J. Hirschberg. 2007. V-measure: A conditional entropy-based external cluster evaluation measure. In *EMNLP-CoNLL*, Vol. 7. 410–420.

J. Sang and C. Xu. 2011. Browse by chunks: Topic mining and organizing on web-scale social media. *ACM Transactions on Multimedia Computing, Communications, and Applications* 7, 1 (2011), 30.

C. J. Sigrist, E. De Castro, L. Cerutti, B. A. Cuche, N. Hulo, A. Bridge, L. Bougueleret, and I. Xenarios. 2013. New and continuing developments at PROSITE. *Nucleic acids research* 41, D1 (2013), D344–D347.

N. X. Vinh, J. Epps, and J. Bailey. 2010. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *The Journal of Machine Learning Research* 9999 (2010), 2837–2854.

X. Wan. 2007. A novel document similarity measure based on earth mover's distance. *Information Sciences* 177, 18 (2007), 3718–3730.

D. Xu and S. F. Chang. 2008. Video event recognition using kernel methods with multilevel temporal alignment. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30, 11 (2008), 1985–1997.

W. Zhao, H. Ma, and Q. He. 2009. Parallel k-means clustering based on mapreduce. In *Cloud Computing*. 674–679.

Q. Zheng and W. Gao. 2008. Constructing visual phrases for effective and efficient object-based image retrieval. *ACM Transactions on Multimedia Computing, Communications, and Applications* 5, 1 (2008), 7.

Received May 2014; revised Sep 2014; accepted Dec 2014

Online Appendix to: Parallel Massive Clustering of Discrete Distributions

YU ZHANG, JAMES Z. WANG and JIA LI, The Pennsylvania State University

A. RUN-TIME ANALYSIS OF PD2-CLUSTERING

The major operations in the PD2-clustering algorithm is the iterative invocations of the data segmentation and local sequential clustering as described in Section 4.2 and 4.3. Compared to these two operations, the time consumed by the one-time disk access and the data transmission among processors is in fact negligible based on our observation in the experiments. Here we make an overall run-time estimation of the parallel D2-clustering process by considering the time complexities of these two operations. Frequently used notations are listed below:

- N: number of data points (discrete distributions);
- M: number of parallel processors;
- R: local clustering shrinking factor ($R > 2$);
- τ: maximal chunk size for local D2-clustering ($\tau \le N$);
- $f_{D2}(N)$: run-time of sequential D2-clustering;
- $f_C(N)$: run-time of constrained D2-clustering.

For the first pass of the parallel algorithm, the run-time is:

$$T_1(N) = T_{seg}(N) + f_{D2}(\tau)\Theta(\frac{N}{\tau M}) , \qquad (19)$$

where the first term $T_{seg}(N)$ is the time consumed in the data segmentation and the second term is the time for the local clustering.

The initial data segmentation is a process containing a series of binary splitting operations, each of which is done by the constrained D2-clustering algorithm, whose run-time is $f_C(N)$. In practice the segmentation tends to divide the data evenly, therefore $T_{seg}(N)$ follows the recursion:

$$T_{seg}(N) = 2T_{seg}\left(\Theta(\frac{N}{2})\right) + \frac{f_C(N)}{M} . \qquad (20)$$

As shown in Table II, both D2-clustering and constrained D2-clustering are algorithms iteratively updating the data assignment and centroids of clusters in a K-means manner. The essential complexity difference between these two approaches is introduced by different ways of centroid update operations. D2-clustering needs to solve a large LP whose scale is proportional to the data size, which results in a high-order polynomial complexity. The constrained D2-clustering algorithm, on the other hand, decouples the large LP to several small ones with fixed size (only related to the number of supports in each discrete distribution), therefore is theoretically linear in time complexity.

Arthur et al. recently proved that K-means has a smoothed polynomial complexity [Arthur et al. 2009] in terms of the expected number of iterations, which is a more realistic estimation of K-means' time complexity because the worst-case upper bound k^N for the iteration number is rarely met in practice. The analysis is done by bounding the potential (total squared distance) drop in each round of centroid and label updating, which can be also applied in the smoothed analysis of D2-clustering. Therefore we

© 2015 ACM 1551-6857/2015/01-ART1 $15.00
DOI: http://dx.doi.org/10.1145/0000000.0000000

can consider both D2-clustering and constrained D2-clustering converge in polynomial number of iterations, which results in polynomial time complexities, $f_{D2}(N) = \Theta(N^{\alpha})$ and $f_C(N) = \Theta(N^{\beta})$, where $\alpha > \beta > 1$ due to the difference in the centroid update stage. Solving the recursion in Equation (20), we obtain

$$T_{seg}(N) = \frac{\Theta(f_C(N))}{M} , \tag{21}$$

and consequently

$$T_1(N) = \frac{1}{M}\Theta(f_C(N) + \frac{f_{D2}(\tau)}{\tau}N) . \tag{22}$$

The time for the multi-pass process follows the recursion:

$$T(N) = T_1(N) + T(N/R) . \tag{23}$$

The order is leaded by $T_1(N)$ and we deduce that the overall run-time follows the order:

$$T(N) = \Theta(T_1(N)) = \frac{1}{M}\Theta(N^{\beta} + \tau^{\alpha-1}N) . \tag{24}$$

Compared with the time complexity $f_{D2}(N) = \Theta(N^{\alpha})$ for the sequential D2-clustering algorithm, the speed-up of the parallel D2-clustering algorithm comes from two algorithmic improvements. First, the leading order of the time complexity is lowered from N^{α} to N^{β} ($\alpha > \beta$); second, the computation is done in parallel and therefore the overall run-time is divided by M. However, when N is small the parallel algorithm is not necessarily faster because the overhead for parallelization is not negligible.

B. ESTIMATION OF CLUSTER MASS AND DISPERSION

The parallel clustering algorithm has a multi-pass structure. For the first pass, the inputs are N discrete distributions $V_1, V_2, \ldots, V_N$, and k_1 clusters $G_j^{(1)} = \{V_i : L_0^1(i) = j\}, j = 1, \ldots, k_1$ are obtained, where $L_0^1(\cdot)$ is the cluster assignment function in this pass and $Z_j^{(1)}$ is the centroid for $G_j^{(1)}$. For the l-th pass ($l > 1$), the objects being clustered are the cluster centroids obtained from the $(l-1)$-th pass, i.e., $Z_1^{(l-1)}, \ldots, Z_{k_{l-1}}^{(l-1)}$, and the results are k_l clusters $G_j^{(l)} = \{Z_i^{(l-1)} : L_{l-1}^l(i) = j\}, j = 1, \ldots, k_l$ and their centroids $Z_j^{(l)}$, where $L_{l-1}^l(\cdot)$ is the cluster assignment at level l for $Z_i^{(l-1)}$'s. By inheriting the cluster labels of the later passes, we allocate the original data points $V_1, \ldots, V_N$ to different l-th pass clusters by the assignment:

$$L_0^l(i) = L_{l-1}^l\left(L_{l-2}^{l-1}\left(\ldots L_0^1(i)\right)\right), i = 1, 2, \ldots, N . \tag{25}$$

After the clustering in each pass, we need to estimate the mass and dispersion of each cluster to determine whether the clustering process should stop. To avoid imposing too much computation overhead to the algorithm, evaluation of these properties needs to be fast. Instead of retrieving back to the original input data, we recursively perform the evaluation.

As introduced in Section 4.3, the weight of each intermediate centroid to be clustered in the l-th pass is in fact the mass of the corresponding cluster. Denote $\omega_j^{(l-1)}$ as the weight of centroid $Z_j^{(l-1)}$ ($l > 1$), which represents the cluster $G_j^{(l-1)}$ in the $(l-1)$-th pass. $\omega_j^{(l-1)}$ is also the mass of cluster $G_j^{(l-1)}$. Because $\omega_j^{(l-1)}$ are passed together with $Z_j^{(l-1)}$ to the l-th pass, the cluster mass in the l-th pass can be directly evaluated by

$$\omega_j^{(l)} = \sum_{i \in \Phi_{l-1}^l(j)} \omega_i^{(l-1)}, j = 1, \ldots, k_{l-1} , \tag{26}$$

where $\Phi_{l-1}^{l}(j) = \{i : L_{l-1}^{l}(i) = j\}$ is the set containing the indices of all $(l-1)$-th pass data points belonging to the cluster $G_j^{(l)}$ in the l-th pass.

If the cluster dispersions in the $(l-1)$-th pass are known, we can estimate the within-cluster dispersion $\Delta_j^{(l)}$ based on the triangle inequality of the Mallows distance (which is proved to be a metric [Clement and Desch 2008]):

$$\begin{aligned}\Delta_j^{(l)} =& \frac{1}{\omega_j^{(l)}} \sum_{i\in\Phi_0^l(j)} D^2(V_i, Z_j^{(l)}) \le \frac{1}{\omega_j^{(l)}} \sum_{i\in\Phi_0^l(j)} \left(D(V_i, Z_{L_0^{l-1}(i)}^{(l-1)}) + D(Z_{L_0^{l-1}(i)}^{(l-1)}, Z_j^{(l)})\right)^2 \\ \le& \frac{1}{\omega_j^{(l)}} \sum_{j'\in\Phi_{l-1}^l(j)} \omega_{j'}^{(l-1)} \left(\Delta_{j'}^{(l-1)} + D^2(Z_{j'}^{(l-1)}, Z_j^{(l)}) + 2D(Z_{j'}^{(l-1)}, Z_j^{(l)})\sqrt{\Delta_{j'}^{(l-1)}}\right) ,\end{aligned} \tag{27}$$

where the last inequality is derived by the fact that for a set of values $\{D(V_i, Z_{j'}^{(l-1)}) : i \in \Phi_0^{l-1}(j')\}$ the mean is smaller than or equal to the root mean square, i.e.,

$$\frac{1}{\omega_{j'}^{(l-1)}} \sum_{i\in\Phi_0^{l-1}(j')} D(V_i, Z_{j'}^{(l-1)}) \le \sqrt{\frac{1}{\omega_{j'}^{(l-1)}} \sum_{i\in\Phi_0^{l-1}(j')} D^2(V_i, Z_{j'}^{(l-1)})} \ .$$

In the l-th pass, the term $D(Z_{j'}^{(l-1)}, Z_j^{(l)})$ in (27) is computed when assigning the cluster labels, $L_{l-1}^{l}(\cdot)$. The dispersion for each cluster $\Delta_{j'}^{(l-1)}$ from the $(l-1)$-th pass is transmitted to the corresponding processor in the l-th pass together with the centroid $Z_{j'}^{(l-1)}$ and its weight $\omega_{j'}^{(l-1)}$ in the PD2-clustering algorithm. As a result, an upper bound of the within-cluster dispersion for each cluster in the l-th pass can be efficiently estimated without any extra computation of the distances between V_i and $Z_{L_0^l(i)}^{(l)}$.

C. APPROXIMATION BOUND OF PD2-CLUSTERING

The multi-pass clustering is an iterative process that keeps abstracting the dataset by a set of representatives. In each pass of the data reduction, some information is discarded because each cluster of data points are represented by a single cluster centroid. The clustering optimization in the following pass only depends on the representative centroids, therefore is an approximation of the clustering on the original dataset.

Intuitively data in a tight cluster can be well represented by the centroid. For a dataset $\{V_1, \ldots, V_N\}$ and a partition that divides the dataset into subsets $\chi_1, \chi_2, \ldots, \chi_{\mathscr{L}}$, the data in each subset are sent to different parallel processors and optimized separately to find tight clusters inside the partition. With a fixed shrinking factor R (i.e., the average number of data points in each cluster), the closeness of data points inside a partition affects the tightness of the clusters in it. A location-based partitioning strategy as introduced in Section 4.2 is therefore beneficial to the generation of tight intermediate clusters that can reduce the approximation error.

Assume PD2-clustering has l passes, and k_γ centroids are obtained in the γ-th pass. D2-clustering result with k_l centroids on the same dataset has a cost function $\psi^* = \sum_{j=1}^{k_l} \sum_{L^*(i)=j} D^2(V_i, Z_j^*)$, where $L^*(i)$ is the optimal cluster assignment and Z_j^*'s are the optimal centroids. The centroids $Z_j^{(l)}$'s and cluster assignment L_0^l obtained from the l-th pass result in the clustering cost of:

$$\psi^l = \sum_{j=1}^{k_l} \sum_{L_0^j(i)=j} D^2(V_i, Z_j^{(l)}) \ . \tag{28}$$

Substituting Equation (27) into the cost, we get

$$
\begin{aligned}
\psi^* \leq & \psi^l = \sum_{j=1}^{k_l} \sum_{L_0^j(i)=j} D^2(V_i, Z_j^{(l)}) = \sum_{j=1}^{k_l} \omega_j^{(l)} \Delta_j^{(l)} \\
\leq & \sum_{j=1}^{k_l} \sum_{j' \in \Phi_{l-1}^l(j)} \omega_{j'}^{(l-1)} \left(\Delta_{j'}^{(l-1)} + D^2(Z_{j'}^{(l-1)}, Z_j^{(l)}) + 2\sqrt{\Delta_{j'}^{(l-1)}} D(Z_{j'}^{(l-1)}, Z_j^{(l)}) \right) \\
= & \sum_{j'=1}^{k_{l-1}} \omega_{j'}^{(l-1)} \left(\Delta_{j'}^{(l-1)} + D^2(Z_{j'}^{(l-1)}, Z_{L_{l-1}^l(j')}^{(l)}) + 2\sqrt{\Delta_{j'}^{(l-1)}} D(Z_{j'}^{(l-1)}, Z_{L_{l-1}^l(j')}^{(l)}) \right) \\
\leq & \sum_{j'=1}^{k_{l-1}} 2\omega_{j'}^{(l-1)} \left(\Delta_{j'}^{(l-1)} + D^2(Z_{j'}^{(l-1)}, Z_{L_{l-1}^l(j')}^{(l)}) \right) \\
= & 2\psi^{l-1} + 2 \sum_{j'=1}^{k_{l-1}} \omega_{j'}^{(l-1)} D^2(Z_{j'}^{(l-1)}, Z_{L_{l-1}^l(j')}^{(l)}) \ .
\end{aligned}
\tag{29}
$$

Recursively applying the inequality, we obtain the approximation bound:

$$
\psi^* \leq \psi^l \leq \sum_{\gamma=0}^{l-1} \left(2^{l-\gamma} \sum_{j=1}^{k_\gamma} \omega_j^{(\gamma)} D^2(Z_j^{(\gamma)}, Z_{L_\gamma^{\gamma+1}(j)}^{(\gamma+1)}) \right) , \tag{30}
$$

where the original data $V_j = Z_j^{(0)}$ and $\omega_j^{(0)} = 1, \forall j \in \{1, \ldots, N\}$.

Notice that in Equation (30), term $\sum_{j=1}^{k_\gamma} \omega_j^{(\gamma)} D^2(Z_j^{(\gamma)}, Z_{L_\gamma^{\gamma+1}(j)}^{(\gamma+1)})$ is the optimization objective function for the $(\gamma+1)$-th pass (denoted as σ_γ) . σ_γ is amplified by the factor $2^{l-\gamma}$ when accumulating to the total clustering cost with respect to the original data points as described in Equation (28). The bound given by Equation (30) validates and explains our intuition to partition the dataset by data locations. To lower the upper bound for ψ^l, the clustering costs in the earlier passes need to be small. Hence we greedily apply the location-based data partitioning from the first pass of the clustering, aiming to keep each local cluster tight from the beginning of the process.

We analyze the impact of R and τ based on three facts as observed in the experiment in Section 5.2. (1) The data being clustered at different passes have similar distributions. (2) In the $(\gamma+1)$-th pass, σ_γ is an decreasing function of the total cluster number $N/R^{(\gamma+1)}$ in a polynomial order given τ fixed. (3) In the $(\gamma+1)$-th pass, σ_γ is a increasing function of the number of chunks $N/(R^\gamma \tau)$ in a polynomial order given R fixed.

Based on the above observations we have $\sigma_\gamma = \psi^*(N/R^{(\gamma+1)})^\kappa(N/(R^\gamma\tau))^\lambda$, where $\kappa < 0$ and $\lambda > 0$ are constants. Expanding σ_γ in Equation (30), we get:

$$
\psi^* \leq \psi^l \leq \psi^* N^{\kappa+\lambda} \tau^{-\lambda} R^{-\kappa} \sum_{\gamma=0}^{l-1} 2^{l-\gamma} R^{-\gamma(\kappa+\lambda)} \ . \tag{31}
$$

This equation estimates the approximation bound of PD2-clustering with respect to sequential D2-clustering. Clearly a large τ is desired to get a tight bound. Because $l = \lceil \log_R \frac{N}{k_l} \rceil$ (k_l is the cluster number), increasing R tends to reduce the summation term in Equation (31) [10] but increase the value of $R^{-\kappa}$. Though κ and λ are unknown

[10]Based on the experiment, $\kappa + \lambda \simeq 1$. Both the term values and term numbers of the summation decrease with R. l is not continuous with R. With same l, it is better use a smaller R for a tighter bound.

and there is no analytical solution of the optimal value, we can tell that R should be neither too large nor too small. Empirical values of R and τ are given in Section 5.2.

D. SYNTHETIC DATASETS AND CLUSTERING EVALUATION CRITERIA

In order to quantitatively evaluate the performance of PD2-clustering, we adopt synthetic data with known cluster labels as the ground-truth in the experiments.

In the Euclidean space where data points are vectors, the standard way to generate synthetic data for clustering evaluation is to randomly sample some discrete points as cluster centroids first and then sample data points around each centroid from a Gaussian distribution with certain covariance. We adopt a similar strategy to generate discrete distributions for D2-clustering. Suppose we want the synthetic discrete distributions to form k clusters, and the supports for each discrete distribution lie in a d dimensional vector space, we first randomly sample k vectors, $z_1, z_2, \ldots, z_k \in \mathscr{R}^d$, and use the distributions with a single support $Z_j = \{(z_j, 1)\}, j = 1, \ldots, k$ as the centroids. A discrete distribution V_i around the centroid Z_j is composed of t support vectors $v_i^{(1)}, \ldots, v_i^{(t)}$ that follow a Gaussian distribution centered at z_j, with their weights $p_i^{(1)}, \ldots, p_i^{(t)}$ randomly sampled from a Dirichlet distribution. With properly tuned variance of the support vectors, data points belonging to different clusters will be slightly overlapped but in general gathering around the cluster centroids. In our experiments, we set the dimension of support vectors $d = 3$, and generate $k = 10$ clusters of data points. Synthetic sets with sizes N from 500 to 1,024,000 are generated by the same distributions in order to benchmark different algorithms' performances in different scales. About 7 percent of data in the synthetic sets are overlapped, i.e., they are closer to other centroids than the one they are sampled from.

With the ground-truth label for each data point, the clustering quality can be evaluated by some *external criteria*, including Adjusted Rand Index (ARI) [Hubert and Arabie 1985], normalized mutual information (NMI) [Vinh et al. 2010], Homogeneity, Completeness, and V-measure [Rosenberg and Hirschberg 2007]. In our experiments, these evaluation metrics changes in similar trend in different clustering runs. For simplicity and clarity, we report ARI in this paper as the clustering quality benchmark. On a certain dataset, assume the ground-truth divides the data points into groups $\{G_1, \ldots, G_m\}$ and the clustering algorithm gets a partition $\{G'_1, \ldots, G'_{m'}\}$, where $m_i = |G_i|$, $m'_j = |G'_j|$, and $N_{i,j} = |G_i \cap G'_j|$, the ARI is defined as:

$$ARI = \frac{\sum_{ij} \binom{N_{i,j}}{2} - \left[\sum_i \binom{m_i}{2} \sum_j \binom{m'_j}{2}\right] / \binom{N}{2}}{\frac{1}{2}\left[\sum_i \binom{m_i}{2} + \sum_j \binom{m'_j}{2}\right] - \left[\sum_i \binom{m_i}{2} \sum_j \binom{m'_j}{2}\right] / \binom{N}{2}} \,. \tag{32}$$

ARI ranges from 0 to 1. Higher ARI indicates more similarity between the ground-truth and the cluster labels ($ARI = 1$ if they are identical).

The clustering objective function, i.e., the total within-cluster dispersions, can serve as an *internal criterion* of the clustering quality. It can be evaluated without the ground-truth, and is an intrinsic property regarding the cluster tightness (smaller dispersion means tighter clusters). To compare the cluster tightness for different datasets, we use the average dispersion, i.e., the average squared Mallows distance (ASD) from each data point to its closest cluster centroid, as the evaluation. It is formulated as:

$$ASD = \frac{1}{N} \sum_{i=1}^{N} D^2(V_i, Z_{L(i)}) \,, \tag{33}$$

where $D(\cdot, \cdot)$ is the Mallows distance, $L(i)$ is the cluster label of the data point V_i and $Z_{L(i)}$ is the corresponding cluster centroid.