Pattern Recognition 165 (2025) 111654

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

Stochastic limited memory bundle algorithm for clustering in big data

Napsu Karmitsa [a,*], Ville-Pekka Eronen [a], Marko M. Mäkelä [b], Tapio Pahikkala [a], Antti Airola [a]

[a] Department of Computing, University of Turku, Finland
[b] Department of Mathematics and Statistics, University of Turku, Finland

ARTICLE INFO

Dataset link: https://github.com/napsu/bigClust

Keywords:
Clustering
Nonsmooth optimization
Nonconvex optimization
Stochastic gradient
Limited memory bundle method

ABSTRACT

Clustering is a crucial task in data mining and machine learning. In this paper, we propose an efficient algorithm, Big-Clust, for solving minimum sum-of-squares clustering problems in large and big datasets. We first develop a novel stochastic limited memory bundle algorithm (SLMBA) for large-scale nonsmooth finite-sum optimization problems and then formulate the clustering problem accordingly. The Big-Clust algorithm — a stochastic adaptation of the incremental clustering methodology — aims to find the global or a high-quality local solution for the clustering problem. It detects good starting points, i.e., initial cluster centers, for the SLMBA, applied as an underlying solver. We evaluate Big-Clust on several real-world datasets with numerous data points and features, comparing its performance with other clustering algorithms designed for large and big data. Numerical results demonstrate the efficiency of the proposed algorithm and the high quality of the found solutions on par with the best existing methods.

1. Introduction

Clustering is one of the most important tasks in data mining and machine learning. It deals with the problem of organizing and grouping similar data points together based on inherent patterns or features. Clustering is used, for instance, in medicine and bioinformatics [2–4], cyber security [5,6], text mining [7], and image processing [8], as well as in the fields of economy, finance, and marketing [9].

In this paper, we focus on the *hard clustering problem*, wherein each data point belongs exclusively to one cluster: data points within the same cluster are somewhat similar to each other and dissimilar to the data points in other clusters. The similarity measure is a key notion in cluster analysis and can be defined using different norms. The squared Euclidean norm is the most common choice as a similarity measure, and clustering problems utilizing this norm are known as *the minimum sum-of-squares clustering (MSSC) problem*.

The MSSC problem can be formulated as a global optimization problem, and various optimization techniques have been employed in designing clustering algorithms. These techniques include

- nonsmooth optimization-based clustering algorithms [10–13];
- methods based on the difference of convex (DC) representation of the clustering function [11,14–17];
- hyperbolic smoothing techniques [18,19]; and
- metaheuristics such as simulated annealing [20,21], tabu search [22,23], particle swarm optimization [2,24], and genetic algorithms [25,26].

In addition, heuristics such as the k-means algorithm and its variations are widely used to solve the MSSC problem (see, e.g., [1,27–32]). It is worth noting that the MSSC problem is NP-hard [33], and solving it is a challenging task. An exact algorithm for solving MSSC problem is given in [34].

Nowadays, computer hardware allows us to store datasets containing millions of data points with large numbers of features in the RAM. This creates a possibility to consider a huge amount of data at once and to achieve accurate clustering results in such datasets. However, most existing clustering algorithms are not applicable in such datasets since either they can only generate suboptimal solutions or they require a prohibitively large computational effort. It is, therefore, imperative to develop algorithms that can generate accurate solutions to clustering problems in very large and big[1] datasets in a reasonable time.

Nonsmooth optimization (NSO) refers to the general problem of minimizing or maximizing functions with discontinuous gradients [35]. When using the nonsmooth formulation of the clustering problem [11],

* Corresponding author.
E-mail address: napsu@karmitsa.fi (N. Karmitsa).

[1] We use here a definition of big data given in [1]: big data is a dataset of such a massive volume that its processing causes significant technical difficulties or is impossible when using traditional methods and average computing resources.

https://doi.org/10.1016/j.patcog.2025.111654
Received 28 June 2024; Received in revised form 20 February 2025; Accepted 26 March 2025
Available online 5 April 2025
0031-3203/© 2025 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

the number of variables in the underlying optimization problem does not depend on the number of data points (m) but only on the number of features (n) and clusters (k). Especially in large datasets, this is a clear advantage of the nonsmooth formulation compared to the more commonly used mixed integer programming formulation, which, in turn, has mk binary and nk continuous variables.

An attempt to address large clustering problems through the application of nonsmooth formulations is made in [13] (also refer to [16] for a slightly different approach). In [13], an incremental algorithm developed by Ordin and Bagirov [36] is combined with the limited memory bundle method (LMBM) proposed by Karmitsa (née Haarala) et al. [37,38] to yield an efficient deterministic clustering algorithm, denoted as LMB-Clust.

In this paper, we improve the LMB-Clust algorithm by introducing its stochastic counterpart, Big-Clust, tailored for addressing very large and big MSSC problems.[2] The proposed method consists of two novel algorithms: the main algorithm represents a *stochastic* modification of the *incremental algorithm* that is used to generate appropriate initial cluster centers, while the *stochastic limited memory bundle algorithm* (SLMBA), a stochastic mini-batch version of the LMBM, is introduced and used to solve the underlying optimization problems. As a result, Big-Clust can be characterized as "doubly stochastic", as it incorporates stochasticity in both the incremental algorithm and the optimization process. This novel doubly stochastic approach enables Big-Clust to handle larger problems more effectively than the deterministic LMB-Clust algorithm, thereby significantly enhancing its applicability and performance.

This paper makes several significant contributions:

1. We introduce a novel SLMBA for effectively solving large-scale nonsmooth finite-sum optimization problems.
2. We present a highly efficient doubly stochastic Big-Clust algorithm tailored for solving clustering problems in datasets containing millions of data points and hundreds of features.
3. We conduct a numerical evaluation of the proposed Big-Clust algorithm and compare its performance in very large-scale and big datasets with the LMB-Clust [13], HG-Means [25], Big-Means [1], DRS-Means [30], DC-Clust [14], MS-MGKM [36], and MiniBatchKMeans from Scikit-learn version 1.5.2 (stable) [39].
4. We offer an open-source implementation of Big-Clust at https://github.com/napsu/bigClust.

The paper is divided into two main parts. The first part focuses on general NSO and introduces the SLMBA (Sections 2 and 3), while the second part is dedicated to clustering and presents the Big-Clust algorithm (Sections 4–6).

2. Preliminaries

We start by introducing the most important aspects of NSO and a finite-sum optimization problem. The notations used are summarized in Table 1.

Nonsmooth optimization. In what follows, we denote by $\mathbb{R}^N$ the N-dimensional Euclidean space and by $\boldsymbol{x}^\top \boldsymbol{y} = \sum_{i=1}^N x_i y_i$ the inner product of vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^N$. The associated Euclidean norm is denoted by $\|\boldsymbol{x}\| = (\boldsymbol{x}^\top \boldsymbol{x})^{1/2}$.

A function $f : \mathbb{R}^N \to \mathbb{R}$ is *nonsmooth* if there are one or more points $\boldsymbol{x} \in \mathbb{R}^N$ such that f fails to be continuously differentiable at these points [35]. A function $f : \mathbb{R}^N \to \mathbb{R}$ is *locally Lipschitz continuous* (LLC) on $\mathbb{R}^N$ if for any bounded subset $X \subset \mathbb{R}^N$ there exists $L > 0$ such that

$$|f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq L\|\boldsymbol{x} - \boldsymbol{y}\|_2 \quad \text{for all } \boldsymbol{x}, \boldsymbol{y} \in X.$$

[2] We assume that the dataset fits into RAM. However, with minor modification, Big-Clust works also with datasets that do not fit into RAM completely.

Table 1
Notations for optimization.

N	Number of variables
$\boldsymbol{x} \in \mathbb{R}^N$	Vector of variables
f	Objective function for optimization
$\nabla f(\boldsymbol{x})$	Gradient of f at $\boldsymbol{x}$
$\partial f(\boldsymbol{x})$	Subdifferential of f at $\boldsymbol{x}$
$\boldsymbol{\xi}$	Subgradient, $\boldsymbol{\xi} \in \partial f(\boldsymbol{x})$
M	Number of partial functions
f_i	ith partial function, $i \in \{1 \dots, M\}$
$m_{\mathcal{B}}$	Size of the batch, $m_{\mathcal{B}} \leq M$
$\mathcal{B}$	Set of batch indices, $\mathcal{B} \subseteq \{1, \dots, M\}$, $\lvert\mathcal{B}\rvert = m_{\mathcal{B}}$
$f_{\mathcal{B}}$	Batch function, $f_{\mathcal{B}}(\boldsymbol{x}) = \frac{1}{m_{\mathcal{B}}} \sum_{i \in \mathcal{B}} f_i(\boldsymbol{x})$
$\partial f_{\mathcal{B}}(\boldsymbol{x})$	Stochastic subdifferential, i.e. subdifferential of $f_{\mathcal{B}}$ at $\boldsymbol{x}$
$\boldsymbol{\xi}_{\mathcal{B}}$	Stochastic subgradient, $\boldsymbol{\xi}_{\mathcal{B}} \in \partial f_{\mathcal{B}}(\boldsymbol{x})$
m_c	Number of limited memory corrections
S, U	Limited memory correction matrices $S, U \in \mathbb{R}^{N \times m_c}$

For example, smooth (continuously differentiable) and convex functions always satisfy this property. Whenever a function $f : \mathbb{R}^N \to \mathbb{R}$ is LLC, the *Clarke subdifferential* at a point $\boldsymbol{x} \in \mathbb{R}^N$ can be calculated with the formula [35,40]

$$\partial f(\boldsymbol{x}) = \operatorname{conv}\left\{ \lim_{i \to \infty} \nabla f(\boldsymbol{x}^i) \mid \boldsymbol{x}^i \to \boldsymbol{x} \text{ and } \nabla f(\boldsymbol{x}^i) \text{ exists } \right\},$$

where "conv" is the convex hull of a set and each vector $\boldsymbol{\xi} \in \partial f(\boldsymbol{x})$ is called a *subgradient*. By using this subdifferential, it is easy to derive a necessary optimality condition for $\boldsymbol{x}^* \in \mathbb{R}^N$ to be a local optimizer, namely, $\mathbf{0} \in \partial f(\boldsymbol{x}^*)$. Any point satisfying this condition is called *stationary*. Note that stationarity can also be a sufficient condition for global optimality, for instance, when the considered function is convex.

Finite-sum optimization problem. We consider the problem of minimizing an objective function that has the form of a sum:

$$f(\boldsymbol{x}) = \frac{1}{M} \sum_{i=1}^{M} f_i(\boldsymbol{x}) \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^N$ and f_i $(i = 1, \dots, M)$ are LLC partial functions not necessarily smooth. We can define a stochastic approximation of the objective f as

$$f_{\mathcal{B}}(\boldsymbol{x}) = \frac{1}{m_{\mathcal{B}}} \sum_{i \in \mathcal{B}} f_i(\boldsymbol{x}),$$

where $\mathcal{B} \subseteq \{1, \dots, M\}$ is a set of randomly selected batch indices and $m_{\mathcal{B}} \leq M$ is the size of the batch. That is, $|\mathcal{B}| = m_{\mathcal{B}}$. We call the function $f_{\mathcal{B}}$ a *batch function* and its subdifferential $\partial f_{\mathcal{B}}$ a *stochastic subdifferential*. Each component $\boldsymbol{\xi}_{\mathcal{B}} \in \partial f_{\mathcal{B}}(\boldsymbol{x})$ is called a *stochastic subgradient*.

3. Stochastic limited memory bundle algorithm

In this section, we propose a novel SLMBA for minimizing the large-scale nonsmooth finite-sum function (1). The backbone of the new method is the LMBM [37,38] developed for general large-scale NSO. The LMBM is a deterministic bundle-type method (see, e.g., [41]) that computes the search direction by the limited memory variable metric approach [42]. Thus, the LMBM avoids solving the time-consuming quadratic direction finding problem appearing in standard bundle methods (see, e.g., [41,43]) as well as storing and manipulating large matrices as is the case in the standard variable metric (quasi-Newton) methods. These aspects make the LMBM suitable for solving large-scale NSO problems. Namely, the number of operations needed to calculate the search direction is only linearly dependent on the number of variables.

The proposed SLMBA accelerates the solution process further by considering batch functions $f_{\mathcal{B}}$ with $m_{\mathcal{B}} \leq M$ instead of the whole objective f. The basic idea of the SLMBA is straightforward: within a randomly selected batch $\mathcal{B}$ of size $m_{\mathcal{B}} \leq M$, we apply similar procedures as in LMBM, but with the objective function f and the subgradient

Algorithm 1: SLMBA

Input: A starting point $\boldsymbol{x}^1 \in \mathbb{R}^N$, the batch size $m_{\mathcal{B}} \leq M$, the termination limit $i_{term} > 1$, the iteration limit $i_{\max} > 0$, the number of stored correction vectors $m_c \geq 3$, and a finite-sum function f to be minimized.

Output: A heuristic minimizer $\boldsymbol{x}_{best} \in \mathbb{R}^N$ of f.

Step 0. *Initialization:* Set $f_{best} \leftarrow \infty$ and $h \leftarrow 1$. Initialize the correction matrices S^1 and U^1 as empty matrices.

Step 1. *Updating the best solution:* Compute $f(\boldsymbol{x}^h)$. If $f(\boldsymbol{x}^h) < f_{best}$, set $f_{best} \leftarrow f(\boldsymbol{x}^h)$, $\boldsymbol{x}_{best} \leftarrow \boldsymbol{x}^h$ and $i_{best} \leftarrow 0$. Else, set $i_{best} \leftarrow i_{best} + 1$.

Step 2. *Stopping criterion:* If $i_{best} > i_{term}$ STOP the algorithm with $\boldsymbol{x}_{best}$ as a best solution.

Step 3. *Selection of the batch:* Randomly select an index set $\mathcal{B}_h \subseteq \{1, \ldots, M\}$, with $|\mathcal{B}_h| = m_{\mathcal{B}}$.

Step 4. *Solving the batch problem:* Starting from the point $\boldsymbol{x}^h$ apply the LMBM with saved correction matrices S^h and U^h to minimize the batch function $f_{\mathcal{B}_h}$. Stop the LMBM after $i_{\max}$ iterations or with the stationary point of $f_{\mathcal{B}_h}$. Denote the solution by $\boldsymbol{x}^{h+1}$ and save at most m_c most recent correction vectors to S^{h+1} and U^{h+1}. Set $h \leftarrow h + 1$ and go to Step 1.

$\boldsymbol{\xi} \in \partial f(\boldsymbol{x})$ at a point $\boldsymbol{x} \in \mathbb{R}^N$ replaced by the batch function $f_{\mathcal{B}}$ and the stochastic subgradient $\boldsymbol{\xi}_{\mathcal{B}} \in \partial f_{\mathcal{B}}(\boldsymbol{x})$. A key distinction is that in SLMBA, the batch is reselected every $i_{\max}$ iterations, whereas LMBM operates on the entire dataset at each step. The parameter $i_{\max} > 0$ is user-specified, and we use $i_{\max} = 10$ in our experiments.

A notable novelty of SLMBA is storing and utilizing information from previous batches. Instead of discarding past computations, SLMBA retains objective function information from previous batches using limited memory variable metric matrices — specifically, by saving and updating correction matrices S and U. These matrices consist of vectors representing the differences between consecutive iteration points and their corresponding stochastic subgradients. We call these vectors 'correction vectors', and we keep in the memory (saved in S and U) the m_c most recent ones. Here, $m_c > 0$ is a user-specified small number, typically $3 \leq m_c \leq 15$. The correction vectors are used inside the LMBM to implicitly compute the limited memory variable metric updates and, thus, the search direction. Therefore, the time needed to calculate the search direction and the storage requirement of the SLMBA is $O(N)$.

We refer to [37,38] for more information on the LMBM — and, therefore, procedures used in SLMBA as well — and here we give an iterative scheme of the proposed SLMBA as Algorithm 1. In addition, we present a step-by-step form of the algorithm (including the LMBM) in Appendix.

Remark 1. Algorithm 1 is heuristic and does not necessarily converge to a stationary point of the objective function f in general. This can be seen with a simple Example 1. The algorithm terminates if the value of the objective f does not improve in i_{term} last batches (in our experiments $i_{term} = 10$).

Example 1. Let $f = \frac{1}{3}\sum_{i=1}^{3} \|\boldsymbol{x} - \boldsymbol{a}_i\|^2$ with $\boldsymbol{x}, \boldsymbol{a}_i \in \mathbb{R}^2$ and $\boldsymbol{a}_1 = (0,0)^\top$, $\boldsymbol{a}_2 = (0,1)^\top$, and $\boldsymbol{a}_3 = (1,0)^\top$. Define partial functions $f_1 = x_1^2 + x_2^2$, $f_2 = x_1^2 + (x_2 - 1)^2$, and $f_3 = (x_1 - 1)^2 + x_2^2$, and apply Algorithm 1 with $m_{\mathcal{B}} = 1$. The algorithm loops between the stationary points $\boldsymbol{x} = \boldsymbol{a}_1$, $\boldsymbol{x} = \boldsymbol{a}_2$, and $\boldsymbol{x} = \boldsymbol{a}_3$ of partial functions f_1, f_2, and f_3, respectively, finally returning the minimum point $\boldsymbol{x}_{best} = \boldsymbol{a}_1 = (0,0)$ of f_1, which gives the smallest value of the original objective: $f_{best} = f(0,0) = \frac{2}{3}$ (see Steps 1 and 2 of Algorithm 1). It does not find the minimum $f(\frac{1}{3}, \frac{1}{3}) = \frac{4}{9}$ of the original objective.

Remark 2. If Algorithm 1 is used with $m_{\mathcal{B}} = M$, it reverts to the original LMBM with few extra iterations before termination. Therefore, with $m_{\mathcal{B}} = M$ Algorithm 1 is globally convergent for LLC functions [38].

Remark 2 can be used to develop a globally convergent version of the SLMBA. We give this modification as Algorithm 2. The basic idea of this algorithm is to first find an approximation of the stationary point by Algorithm 1 with $m_{\mathcal{B}} < M$ and then to fine-tune this solution with the LMBM.

Remark 3. Applying Algorithm 2 may be impractical in many real-world applications when M is large. Nevertheless, using Algorithm 1 alone with reasonable large batch size $m_{\mathcal{B}}$ gives a good enough approximation of the stationary point of the original function f in most cases, as will be shown in our numerical experiments. In addition, the precision with which Algorithm 1 finds the stationary point can be improved by increasing the batch size.

4. Clustering problem

In this section, we present the *NSO formulation of the clustering problem*. Further, we introduce the *auxiliary clustering problem* used to generate appropriate starting points for the actual clustering problem. But first, we give the notations used in clustering in Table 2, and we recall the definitions of the clustering and similarity measures. For more in-depth information, we refer to [11].

Cluster analysis. Let A be a set consisting of a finite number of points in an n-dimensional space $\mathbb{R}^n$, defined as:

$$A = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m\}, \ \boldsymbol{a}_i \in \mathbb{R}^n, \quad i = 1, \ldots, m.$$

Each data point $\boldsymbol{a}_i$, $i = 1, \ldots, m$ has n *features*. In the *unconstrained hard clustering problem*, our objective is to distribute the points from set A into a given number k of separate *clusters* A^j, $j = 1, \ldots, k$ according to the predefined criteria such that

1. $A^j \neq \emptyset$, $j = 1, \ldots, k$;
2. $A^j \bigcap A^l = \emptyset$, for all $j, l = 1, \ldots, k$, $j \neq l$; and
3. $A = \bigcup_{j=1}^{k} A^j$.

Each cluster A^j can be uniquely characterized by its *center* $\boldsymbol{x}_j \in \mathbb{R}^n$, $j = 1, \ldots, k$, and each data point $\boldsymbol{a}_i$, $i = 1, \ldots, m$, belongs to the cluster A^j whose center $\boldsymbol{x}_j$ is the closest one. The problem of finding these centers is called the *k-clustering* (or k-partition) *problem*.

The notion of the *similarity measure* is essential to formulate the clustering problem. There are various similarity functions available (see, e.g., [11]). A common practice is to define a similarity measure as the inverse of a distance metric: the smaller the distance between two points, the more similar they are. Here, we define the similarity measure using the squared Euclidean norm (the L_2-norm)

$$d(\boldsymbol{x}, \boldsymbol{a}) = \sum_{i=1}^{n} (x_i - a_i)^2, \qquad \boldsymbol{x}, \boldsymbol{a} \in \mathbb{R}^n.$$

Clustering problems defined using this similarity measure are called the *minimum sum-of-squares clustering* (MSSC) problems.

Clustering problem. The *NSO formulation of the MSSC* is given by [11, 13]

$$\begin{cases} \text{minimize} & f^k(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) \\ \text{subject to} & (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) \in \mathbb{R}^{nk}, \end{cases} \tag{2}$$

where

$$f^k(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = \frac{1}{m} \sum_{i=1}^{m} \min_{j=1,\ldots,k} d(\boldsymbol{x}_j, \boldsymbol{a}_i). \tag{3}$$

Algorithm 2: SLMBA with global convergence

Input: A starting point $\boldsymbol{x}^1 \in \mathbb{R}^N$, an initial batch size $m_{\mathcal{B}}^{init} \leq M$, and a finite-sum function f to be minimized.

Output: A stationary point $\boldsymbol{x}^* \in \mathbb{R}^N$ of function f.

Step 1. *Heuristic solution:* Starting from point $\boldsymbol{x}^1 \in \mathbb{R}^N$, apply Algorithm 1 with the batch size $m_{\mathcal{B}} = m_{\mathcal{B}}^{init}$ to obtain a heuristic minimizer $\bar{\boldsymbol{x}} \in \mathbb{R}^N$ of function f.

Step 2. *Stationary solution:* Starting from point $\bar{\boldsymbol{x}}$, apply Algorithm 1 with the batch size $m_{\mathcal{B}} = M$ to obtain a stationary point $\boldsymbol{x}^* \in \mathbb{R}^N$ of function f.

Table 2
Notations for clustering.

m	Number of data points
n	Number of features
k	Number of clusters
$\boldsymbol{a}_i \in \mathbb{R}^n$	Data point, $i = 1, \ldots, m$
$A = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m\}$	Dataset
A_S	Uniform random sample of set A
m_S	Size of the random sample A_S, $m_S \leq m$
A^j	Cluster, $j = 1, \ldots, k$
$\boldsymbol{x}_j \in \mathbb{R}^n$	Cluster center, $j = 1, \ldots, k$
$(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) \in \mathbb{R}^{nk}$	Vector of variables for optimization
$f^k(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)$	kth cluster function, $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) \in \mathbb{R}^{nk}$
$\bar{f}^k(\boldsymbol{y})$	kth auxiliary cluster function, $\boldsymbol{y} \in \mathbb{R}^n$
$d(\boldsymbol{x}_j, \boldsymbol{a}_i)$	Squared Euclidean distance, $\boldsymbol{x}_j, \boldsymbol{a}_i \in \mathbb{R}^n$
r_{k-1}^i	Squared distance between $\boldsymbol{a}_i$ and its cluster center in the solution of $(k-1)$-clustering problem
$z^k(\boldsymbol{y})$	Difference $f^k(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k-1}, \boldsymbol{y}) - f^{k-1}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k-1})$, $\boldsymbol{x}_j, \boldsymbol{y} \in \mathbb{R}^n$
$A_0, A_1, B(\boldsymbol{a})$	Subsets of A
$C_1, C_2, S_1, S_2 \subset \mathbb{R}^n$	Sets of candidate cluster centers
$S_3 \subset \mathbb{R}^{nk}$	Set of solutions to clustering problem
$\gamma_1, \gamma_2, \gamma_3$	Threshold parameters $\gamma_1, \gamma_2 \in [0, 1]$, $\gamma_3 \in [1, \infty)$

The objective function f^k is called the *kth cluster function*. It is locally Lipschitz continuous for any k, and nonconvex and nonsmooth for $k > 1$. Problem (2) contains only nk continuous variables $\boldsymbol{x}_j \in \mathbb{R}^n$, $j = 1, \ldots, k$, and this number does not depend on the number of data points m. In case of large datasets, this is a huge advantage of nonsmooth formulation when compared, for instance, to the more commonly used mixed integer formulation, which contains mk integer (binary) and nk continuous variables (see, e.g., [11]). In addition, when we use the stochastic mini-batch approach we can split the computation of the sum in (3) to smaller pieces to accelerate the solution process.

Auxiliary clustering problem. Clustering is a global optimization problem, but conventional global optimization methods are prohibitively time-consuming for solving it in large datasets. Therefore, we need to apply a local search algorithm like SLMBA instead. Nevertheless, the success of local search methods in finding the global minimum or a high-quality local solution (i.e., a solution with the objective value close to the global minimum) to the clustering problem heavily depends on the choice of starting cluster centers. These centers serve as the initial points from which the optimization process begins. Different approaches are used to generate such points, for instance in [11,36,44]. In this paper, we utilize the auxiliary clustering problem [36] for finding a set of "good/promising" starting points for the clustering problem.

Assume that the solution $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k-1})$, $k \geq 2$ to the $(k-1)$-clustering problem is known. Denote by r_{k-1}^i the squared distance between the data point $\boldsymbol{a}_i$, $i = 1, \ldots, m$ and its cluster center

$$r_{k-1}^i = \min_{j=1,\ldots,k-1} d(\boldsymbol{x}_j, \boldsymbol{a}_i). \tag{4}$$

The *kth auxiliary cluster function* is defined as

$$\bar{f}^k(\boldsymbol{y}) = \frac{1}{m}\sum_{i=1}^{m} \min\left\{r_{k-1}^i, d(\boldsymbol{y}, \boldsymbol{a}_i)\right\}, \quad \boldsymbol{y} \in \mathbb{R}^n, \tag{5}$$

and the *auxiliary k-clustering problem* is formulated as

$$\begin{cases} \text{minimize} & \bar{f}^k(\boldsymbol{y}) \\ \text{subject to} & \boldsymbol{y} \in \mathbb{R}^n. \end{cases} \tag{6}$$

Similarly to the clustering problem (2), this problem is both nonsmooth and nonconvex. Nevertheless, it has a simpler structure, less local minima, and only n variables. Moreover,

$$\bar{f}^k(\boldsymbol{y}) = f^k(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k-1}, \boldsymbol{y}) \quad \text{for all } \boldsymbol{y} \in \mathbb{R}^n, \tag{7}$$

and we can compute the difference

$$z^k(\boldsymbol{y}) = \frac{1}{m}\sum_{i=1}^{m} \max\{0, r_{k-1}^i - d(\boldsymbol{y}, \boldsymbol{a}_i)\},$$

which gives us the decrease of the value of the kth cluster function f^k at point $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k-1}, \boldsymbol{y})$ from that of the $(k-1)$th cluster function f^{k-1} at $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k-1})$. We recall a slightly modified algorithm for finding a set of starting points for the auxiliary l-clustering problem (6) ($l = 2, \ldots, k$) [11,36] as Algorithm 3.

Remark 4. Set A_0 (see Step 1 in Algorithm 3) consists of all data points that are not cluster centers while set A_1 contains points that give the decrease of the lth cluster function no less than the threshold $\gamma_1 z_1^{\max}$. Set $B(\boldsymbol{a}^*)$ (see Step 2 in Algorithm 3) consists of all data points $\boldsymbol{a}_i \in A$ that are closer to $\boldsymbol{a}^* \in A_1$ than to their current cluster center $\boldsymbol{x}_j$, $j \in \{1, \ldots, l-1\}$. Set C_1 contains the centers $\boldsymbol{c}(\boldsymbol{a}^*)$ for all $\boldsymbol{a}^* \in A_1$, and set C_2 is obtained from set C_1 by removing centers that do not provide the sufficient decrease of the lth cluster function (sufficient decrease is defined by the threshold $\gamma_2 z_2^{\max}$). Since we use all data points from set A to compute set C_2, it contains starting points from different parts of the dataset A.

Remark 5. Threshold parameters γ_1 and γ_2 define the proportion of points (A_1) selected for Step 2 of Algorithm 3 and the final set of starting points (C_2), respectively. When $\gamma_i = 0$ ($i = 1, 2$), all points are selected while $\gamma_i = 1$ gives only the best points (the ones that decrease the l-cluster function value the most). For large datasets, one can choose e.g., $\gamma_1 = 0.95$ and $\gamma_2 = 0.99$.

Remark 6. In the case of big data (say $m > 100000$), Algorithm 3 can be applied to a uniform random sample A_S of size $m_S \ll m$ of the set A.

We consider all points in set C_2 as initial points for the auxiliary clustering problem (6). A local search algorithm, such as SLMBA, generates solutions corresponding to each starting point. Consequently, by considering Eq. (7), solving problem (6) provides us with "good/promising" starting points for the clustering problem (2).

5. Big-clust

Now, we present an incremental clustering algorithm Big-Clust for solving the clustering problem (2). The algorithm starts with one cluster and adds a new cluster center at each iteration until the user-specified number of clusters k is achieved. In addition to the k-clustering problem, Big-Clust solves also all intermediate l-clustering problems where $l = 1, \ldots, k-1$.

As we pointed out in Remark 6, Algorithm 3 can be applied to a uniform random sample A_S of size $m_S \ll m$ instead of the full dataset A in case of very large-scale and big data. The same idea is extended to the entire Big-Clust algorithm (Algorithm 4). At each iteration of the incremental algorithm (i.e., for solving each l-clustering problem with

Algorithm 3: Finding set of starting points

Input: Dataset A, the current number of clusters $l \geq 2$, the solution $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{l-1})$ to the $(l-1)$-clustering problem, and the threshold values $\gamma_1, \gamma_2 \in [0, 1]$.

Output: Set $C_2 \subset \mathbb{R}^n$ of starting points for the auxiliary l-clustering problem and the best center $\boldsymbol{c} \in \mathbb{R}^n$ such that $z^l(\boldsymbol{c}) = z_2^{\max}$.

Step 1. *Selecting data points as initial centers:* Compute $z_1^{\max} = \max_{\boldsymbol{a}' \in A_0} z^l(\boldsymbol{a}')$, where $A_0 = \{\boldsymbol{a}' \in A \mid \boldsymbol{a}' \neq \boldsymbol{x}_i, i = 1, \ldots, l-1\}$. Set $A_1 \leftarrow \{\boldsymbol{a}^* \in A_0 \mid z^l(\boldsymbol{a}^*) \geq \gamma_1 z_1^{\max}\}$.

Step 2. *Computing new centers:* For each $\boldsymbol{a}^* \in A_1$ compute the set $B(\boldsymbol{a}^*) = \{\boldsymbol{a}_i \in A \mid r_{l-1}^i > d(\boldsymbol{a}^*, \boldsymbol{a}_i)\}$ and its center $\boldsymbol{c}(\boldsymbol{a}^*) = \frac{1}{|B(\boldsymbol{a}^*)|} \sum_{\boldsymbol{a}^\dagger \in B(\boldsymbol{a}^*)} \boldsymbol{a}^\dagger$.

Step 3. *Selecting "the best" centers as starting cluster centers:* Compute $z_2^{\max} = \max_{\boldsymbol{c} \in C_1} z^l(\boldsymbol{c})$, where $C_1 = \{\boldsymbol{c} \in \mathbb{R}^n \mid \exists\, \boldsymbol{a}^* \in A_1, \text{ s.t. } \boldsymbol{c} = \boldsymbol{c}(\boldsymbol{a}^*)\}$. Set $C_2 \leftarrow \{\boldsymbol{c}' \in C_1 \mid z^l(\boldsymbol{c}') \geq \gamma_2 z_2^{\max}\}$.

Algorithm 4: Big-Clust

Input: Dataset A, the maximum number of clusters $k \geq 1$, the sample size $m_S \leq m$, the batch size $m_B \leq m_S$, and the threshold value $\gamma_3 \in [1, \infty)$.

Output: The solution $\boldsymbol{x}_j$, $j = 1, \ldots, k$ to the k-clustering problem and all the intermediate solutions with $l = 1, \ldots, k-1$.

Step 0. *Initialization:* Randomly select m_S data points from A. Denote this sample set by A_S. Compute the center $\boldsymbol{x}_1 \in \mathbb{R}^n$ of set A_S. Set $l \leftarrow 1$.

Step 1. *Stopping criterion:* Set $l \leftarrow l + 1$. If $l > k$, STOP: the k-clustering problem has been solved.

Step 2. *Computation of the set of starting points for auxiliary problem:* Apply Algorithm 3 with $A = A_S$ to find the set $C_2 \subset \mathbb{R}^n$ of starting points for the auxiliary l-clustering problem and the point $\boldsymbol{c} \in \mathbb{R}^n$ that gives the biggest decrease of the l-th cluster function value in set A_S.

Step 3. *Solving the auxiliary clustering problem:* Apply the SLMBA (Algorithm 1) with $k = l$ and one randomly selected batch of size m_B to solve problem (6) in A_S starting from each point $\boldsymbol{y} \in C_2$. Denote by $S_1 \subset \mathbb{R}^n$ a set of such solutions.

Step 4. *Selecting new starting cluster centers:* Compute

$$\bar{f}_{\min}^l = \min_{\boldsymbol{y} \in S_1 \cup \{\boldsymbol{c}\}} \bar{f}^l(\boldsymbol{y}).$$

Set $S_2 \leftarrow \{\boldsymbol{y} \in S_1 \cup \{\boldsymbol{c}\} \mid \bar{f}^l(\boldsymbol{y}) \leq \gamma_3 \bar{f}_{\min}^l\}$.

Step 5. *Solving the clustering problem:* Starting from the point $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{l-1}, \bar{\boldsymbol{y}})$ for each $\bar{\boldsymbol{y}} \in S_2$ apply the SLMBA to solve problem (2) in A_S and find a solution $(\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_l)$. Denote by $S_3 \subset \mathbb{R}^{nl}$ a set of all such solutions.

Step 6. *Solution to the l-clustering problem:* Compute

$$f_{\min}^l = \min \left\{ f^l(\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_l) \mid (\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_l) \in S_3 \right\}$$

and the collection of cluster centers $(\bar{\boldsymbol{y}}_1, \ldots, \bar{\boldsymbol{y}}_l)$ such that

$$f^l(\bar{\boldsymbol{y}}_1, \ldots, \bar{\boldsymbol{y}}_l) = f_{\min}^l.$$

Set $\boldsymbol{x}_j \leftarrow \bar{\boldsymbol{y}}_j$, $j = 1, \ldots, l$ as a solution to the l-clustering problem, randomly select m_S data points from A to get a new sample set A_S, and go to Step 1.

$l = 1, \ldots, k$), a new uniformly random sample A_S of size $m_S \leq m$ is selected and clustered with the SLMBA with the batch size $m_B \leq m_S$. However, it is worth noting that clustering in small subsets of data is more prone to all kinds of disruptions in data than clustering in entire data with the stochastic optimization method (see the numerical experiments). Thus, we recommend using $A_S = A$ when possible, considering the number of data points m.

Remark 7. In Step 3 of Algorithm 4, we solve the auxiliary clustering problem in only a part of the data (or part of the data sample A_S): in one batch, but the batch differs for each $\boldsymbol{y}$. Using random batches instead of the entire dataset brings more randomness to the starting point procedure and, therefore, introduces more variability in intermediate solutions. This, in turn, avails in the search for the globally optimal solution. In addition, it provides more efficient computations. However, the solution found for a batch may not decrease the cluster function value in the entire data. Thus, in Step 4, we include the best starting point $\boldsymbol{c}$, provided by Algorithm 3, to the candidate solutions set.

Remark 8. The set S_2 in Step 4 of Algorithm 4 contains points where the value of the function $\bar{f}^l(\boldsymbol{y})$ in no more than the threshold $\gamma_3 \bar{f}_{\min}^l$. Note that the set S_2 is not an empty set: if $\gamma_3 = 1$ it contains the best minimizer of the function $\bar{f}^l(\boldsymbol{y})$ obtained using starting points from the set $S_1 \cup \{\boldsymbol{c}\}$, while with sufficiency large γ_3, $S_2 = S_1 \cup \{\boldsymbol{c}\}$. For large dataset, one can choose e.g., $\gamma_3 = 1.05$.

6. Numerical experiments

We compare the performance of the proposed Big-Clust algorithm with seven other algorithms for large-scale and big data clustering: namely, the original LMB-Clust [13], Big-Means [1], HG-Means [25], DRS-Means [30], DC-Clust [14], MS-MGKM [36], and MiniBatchK-Means from Scikit-learn version 1.5.2 (stable) [39]. In addition, the methodology of our experiments and most of the datasets used are identical to those used in [1,13], allowing additional comparison of our results with those provided in these papers. That includes results obtained with various NSO-based clustering algorithms and with several variants of k-means (see, e.g., [28,45,46]). It is worth noting that, in [13], the LMB-Clust ranked clearly the best method for large-scale clustering while, in [1], Big-Means outperformed the other state-of-the-art algorithms (including the LMB-Clust, k-means++, Forgy k-means, and Ward's algorithm) in big datasets.

Datasets. We evaluate the performances of Big-Clust and competitive algorithms on 22 publicly available real-world datasets. Some details of these sets are provided in Table 3, while more comprehensive descriptions can be found on the corresponding websites listed in Table 4. All datasets contain only numeric features and have no missing values. The number of features (n) ranges from as few as 2 to as many as 5000, while the number of data points (m) varies from thousands (with the smallest dataset containing 7797 points) to over 10 million (with the largest dataset containing 10,500,000 points).

Methods and setup. The implementations of LMB-Clust [13] and the DC optimization-based clustering algorithm, DC-Clust [14], are publicly

Table 3
Brief description of the used datasets.

Dataset	m	n	$m \times n$
ISOLET	7 797	617	4 810 749
Gisette	13 500	5000	67 500 000
Gas sensor array drift	13 910	128	1 780 480
EEG eye state	14 980	14	209 720
D15112	15 112	2	30 224
Online news popularity	39 644	58	2 299 352
KEGG metabolic	53 413	20	1 068 260
Shuttle control	58 000	9	522 000
Sensorless drive diagnosis	58 509	48	2 808 432
MFCCs for speech emotion recognition	85 134	58	4 937 772
Pla85900	85 900	2	171 800
Music analysis	106 574	518	55 205 332
MiniBooNE particle identification	130 064	50	6 503 200
Protein homology	145 751	74	10 785 574
Range queries aggregates	200 000	7	1 400 000
Skin segmentation	245 057	3	735 171
3D road network	434 874	3	1 304 622
Covertype	581 012	10	5 810 120
CORD-19 embeddings[a]	1 056 660	768	811 514 880
US census data 1990	2 458 285	68	167 163 380
BitcoinHeist	2 916 697	8	23 333 576
HEPMASS	10 500 000	28	294 000 000

[a] CORD-19 Embeddings is continuously updated being not the same used in [1].

available at https://napsu.karmitsa.fi/clustering/, while the implementation of the multi-start modified global k-means algorithm, MS-MGKM [36], is obtained directly from the authors. We use the default parameter values provided in these codes for all experiments. The hybrid genetic clustering algorithm, HG-Means [25], is obtained from the code repository https://github.com/danielgribel/hg-means/tree/master. We use the fast configuration of the algorithm with the parameter selection specified in [25].

In addition, the implementation of Big-Means [1], a k-means algorithm for big data clustering, is obtained from the code repository https://github.com/R-Mussabayev/bigmeans. Similarly to [1], we use the Big-Means version with the inner parallelism. That is, Big-Means processes data samples sequentially, but each data sample is clustered using parallelized versions of k-means and k-means++ with *eight* threads. Based on the setup in [1], we select the maximum computational time (in seconds) for Big-Means according to the following rule

$$t_{\max} = \begin{cases} 2, & \text{if } m \times n \leq 1000000 \\ 3, & \text{if } 1000000 < m \times n \leq 10000000 \\ 25, & \text{otherwise.} \end{cases}$$

In addition, for Big-Means the sample size m_S is chosen from two options,

$$m_S^1 = \begin{cases} \min(m-1, 10000), & \text{if } m \leq 50000 \\ 40000, & \text{if } 50000 < m \leq 100000 \\ 60000, & \text{otherwise,} \end{cases}$$

and

$$m_S^2 = \begin{cases} \min(m-1, 10000), & \text{if } m \leq 100000 \\ m/10, & \text{otherwise,} \end{cases}$$

based on the smaller average clustering function value. We apply two different options for selecting the sample size and choose the better one, as Big-Means is quite sensitive to sample size variations. The options given here are based on the averages of m_S values used in [1]. It is worth noting that in [1], the sample size m_S is tuned individually depending on the dataset used: in particular, in Sensorless drive diagnostic, MiniBooNE Particle Identification, and EEG Eye State, the 'entire number of data points minus one' is used. We emphasize that individual sample size selection, although it gives better results, is time-consuming and/or needs preliminary knowledge of the data. Otherwise, the parameters similar to [1] are used with Big-Means. For MiniBatchK-Means we used the default parameters provided in Scikit-learn version 1.5.2 (stable) [39], with the code utilizing parallel computation on *four* cores.

The distributed random swap clustering algorithm, DRS-Means, was implemented based on its description in [30]. Our implementation is available at https://github.com/napsu/DRSmeans. The only tunable parameter in this method is the number of iterations. While the original paper recommends using 10,000 iterations, there is a trade-off between computational efficiency and clustering accuracy. To manage the computational burden in large-scale datasets, we use 1000 iterations, balancing runtime feasibility with effective clustering performance.

Finally, an open-source implementation of the proposed Big-Clust algorithm is given at https://github.com/napsu/bigClust. In our experiments, we use the batch size $m_B = \max\{m/50, 1000\}$, the iteration limit $i_{\max} = 10$, the termination parameter $i_{term} = 10$, and the maximum number of correction vectors $m_c = 7$ with all datasets. In addition, with datasets that have more than 100,000 data points, we use the sample size $m_S = \max\{m/50, 10000\}$ and the batch size $m_B = \max\{m_S/50, 1000\}$. We refer to the latter as Big-Clust+ whenever it is necessary to distinguish between the two versions of the algorithm. In our implementation, parameters γ_1, γ_2, and γ_3 (see Algorithms 3 and 4) are computed implicitly based on more intuitive parameters

- m_{init} — maximum number of data points selected as initial solutions,
- p_1 — percentage of initial points used to form initial centers for the auxiliary clustering problem, and
- p_2 — percentage of auxiliary clustering problem solutions used to form initial centers for the clustering problem,

specified by the user. We use the values $p_1 = 100$, $p_2 = 20$, and

$$5m_{init} = \begin{cases} 500, & \text{if } m \times n \leq 10000000 \\ 200, & \text{otherwise.} \end{cases}$$

Each dataset is clustered ten times with the stochastic algorithm Big-Clust. For HG-Means and DRS-Means, clustering is performed 10 times for each selected k when $m < 100{,}000$ and 5 times when $m > 100{,}000$. The reduced number of runs for larger datasets is chosen because these methods were significantly more time-consuming than the other stochastic algorithms. In addition, for both MiniBatchKMeans and Big-Means, clustering is performed multiple times for each selected k: 10 times for MiniBatchKMeans and 20 times for Big-Means. For Big-Means, more runs are needed due to sample size selection; the sample size that yields the smallest clustering function value averaged over the ten runs is selected. The results with the stochastic algorithms are averaged over the respective runs. The non-stochastic algorithms LMB-Clust, DC-Clust, and MS-MGKM are applied once. The DC-Clust and MS-MGKM algorithms are not designed for clustering very large datasets, let alone big data. Consequently, we have included their results only for datasets with fewer than 100,000 samples.

The Big-Clust, LMB-Clust, and DC-Clust algorithms are implemented in Fortran 95, MS-MGKM is implemented in Fortran 77, Big-Means, DRS-Means, and MiniBatchKMeans in Python, and HG-Means in C++. To ensure the best comparability of the algorithms, regardless of different programming languages, the Python's `time.time()` function is used to measure computational time as it corresponds to Fortran's `cpu_time()` subroutine according to https://github.com/wusunlab/fortran-vs-python. In addition, for Big-Means the time is measured from the beginning of the program similarly to other algorithms, not only inside the sequential loop as in [1]. This fact explains sometimes very different computational times given here compared to those in [1]. In practice, the time used by Big-Means is the time used to read the data plus the maximum allowed CPU time plus some overheads. The computational time of HG-Means is measured with function `clock()` divided by `CLOCKS_PER_SEC`. In all cases, the total time was limited to 10 h per run.

Table 4
Information about the used datasets (date accessed 18.03.2024).

Dataset	URLs
ISOLET	https://archive.ics.uci.edu/dataset/54/isolet
Gisette	https://archive.ics.uci.edu/dataset/170/gisette
Gas sensor array drift	https://archive.ics.uci.edu/dataset/224/gas+sensor+array+drift+dataset
EEG eye state	https://archive.ics.uci.edu/dataset/264/eeg+eye+state
D15112	https://github.com/mastqe/tsplib/blob/master/d15112.tsp
Online news popularity	https://archive.ics.uci.edu/dataset/332/online+news+popularity
KEGG metabolic	https://archive.ics.uci.edu/dataset/220/kegg+metabolic+relation+network+directed
Shuttle control	https://archive.ics.uci.edu/dataset/148/statlog+shuttle
Sensorless drive diagnosis	https://archive.ics.uci.edu/dataset/325/dataset+for+sensorless+drive+diagnosis
MFCCs for speech Emotion recognition	https://www.kaggle.com/datasets/cracc97/features
Pla85900	https://softlib.rice.edu/pub/tsplib/tsp/pla859.tsp.gz
Music analysis	https://archive.ics.uci.edu/dataset/386/fma+a+dataset+for+music+analysis
MiniBooNE particle identification	https://archive.ics.uci.edu/dataset/199/miniboone+particle+identification
Protein homology	https://www.kdd.org/kdd-cup/view/kdd-cup-2004/Data
Range queries aggregates	https://archive.ics.uci.edu/dataset/493/query+analytics+workloads+dataset
Skin segmentation	https://archive.ics.uci.edu/dataset/229/skin+segmentation
3D road network	https://archive.ics.uci.edu/dataset/246/3d+road+network+north+jutland+denmark
Covertype	https://archive.ics.uci.edu/dataset/31/covertype
CORD-19 embeddings	https://www.kaggle.com/datasets/allen-institute-for-ai/CORD-19-research-challenge
US census data 1990	https://archive.ics.uci.edu/dataset/116/us+census+data+1990
BitcoinHeist	https://archive.ics.uci.edu/dataset/526/bitcoinheistransomwareaddressdataset
HEPMASS	https://archive.ics.uci.edu/dataset/347/hepmass

All computational experiments are carried out on iMac, 4.0 GHz Quad-Core Intel(R) Core(TM) i7 machine with 16 GB of RAM. We use gfortran to compile the Fortran codes, Python 3.10 for Python codes with NumPy 1.24.2 and Numba 0.59.0 for Big-Means, and clang++ for HG-Means.

Evaluation metrics and results. Numerical experiment results are summarized in Figs. 1–6, with more detailed results provided in Tables S1–S22 and Figures S1–S22 in the supplementary material. Similarly to [1,13], we use the following metrics to compare different algorithms:

1. *Cluster function values,* also known as the sum of squares error (SSE). The SSE is a prototype-based cohesion measure, showing the average variation over all clusters. In Tables S1–S22, we present the best-known value f_{best} of the cluster function (3), which is to be multiplied by the number shown after the name of the dataset and divided by the number of data points m. We use the f_{best} value given in [1,11,13] or, if not available,[3] the result obtained with LMB-Clust. In CORD-19 Embeddings, in which LMB-Clust fail due to memory issues, we selected the smallest cluster function value obtained in ten separate runs of Big-Means as f_{best}.
   The relative error $E_{\mathcal{A}}$ by an algorithm $\mathcal{A}$ is calculated as
   $$E_{\mathcal{A}} = \frac{\bar{f} - f_{\text{best}}}{f_{\text{best}}} \times 100\%,$$
   where $\bar{f}$ is the value of the cluster function obtained by the algorithm $\mathcal{A}$. For each dataset (and each choice of k in case of Big-Means, HG-Means, DRS-Means, and MiniBatchKMeans), we execute ten (or five) runs, and the minimum ($E_{\min}$), average (E_{aver}), and maximum ($E_{\max}$) errors are reported in the tables. In addition, we give the average of average errors (Aver. E_{aver}) for each method. The average E_{aver} for each dataset are summarized in Figs. 1(a) and 2(a).
2. *Computational time* (in seconds). In Tables S1–S22, we give separately the time used to read the data (t_{init}), the time used for computation of k clusters (t_k), and the total time (t_{total}) used to compute all the clusters in the table. In case of the incremental clustering algorithms Big-Clust, LMB-Clust, DC-Clust, and MS-MGKM, $t_{\text{total}} = t_{init} + t_{25}$, while with Big-Means, HG-Means, DRS-Means, and MiniBatchKMeans $t_{\text{total}} = t_{init} + \sum_l t_l$, where $l = 2, 3, 4, 5, 10, 15, 20, 25$. We summarize the computation times used for $k = 10$ (i.e., t_{10}) in Figs. 1(b) and 2(b) and for $k = 25$ in Figs. 1(c) and 2(c). In addition, we present the variation in computation times relative to the number of clusters in Figs. 3 and 4.
3. *Davies–Bouldin* (DBI) and *Dunn* (DI) *cluster validity indices.* The DBI is given by [47]
   $$\text{DBI} = \frac{1}{k}\sum_{i=1}^{k} \max_{j=1,\ldots,k, i\neq j} \frac{S_k(A^i) + S_k(A^j)}{d(\boldsymbol{x}_i, \boldsymbol{x}_j)},$$
   where $S_k(A^l)$ is the average distance of all data points from the cluster A^l to their cluster center $\boldsymbol{x}_l$, and $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the Euclidean distance between cluster centers $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. The DI is defined as [48]
   $$\text{DI} = \min_{i=1,\ldots,k} \left\{ \min_{j=1,\ldots,k, j\neq i} \left\{ \frac{d(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\max_{l=1,\ldots,k} r(\boldsymbol{x}_l)} \right\} \right\},$$
   where $r(\boldsymbol{x}_l) = \max_{\boldsymbol{a}\in A^l} \|\boldsymbol{x}_l - \boldsymbol{a}\|$ is the radius of the lth cluster.
   The DBI has a small value if the clusters are compact and far away from each other. Consequently, the DBI has the smallest value for optimal clustering. The DI maximizes the inter-cluster distances and minimizes the intra-cluster distances. Therefore, the number of clusters maximizing the DI can be taken as the optimal number of clusters.
   The DBI and DI are computed internally at each iteration of the incremental algorithms and we do not give them to the non-incremental algorithms. Further, for Big-Clust+, these indices are computed only in the current sample set A_S instead of the whole dataset. We illustrate the results in the Gas Sensor Array Drift and US Census Data 1990 datasets in Figs. 5 and 6 , respectively. The results for all datasets are given in Figures S1–S22.

While DC-Clust and MS-MGKM are robust and efficient clustering algorithms for moderate-sized datasets, they are not well-suited for clustering very large datasets or big data due to their high computational burden (see Fig. 1(c)). When comparing average computational times for 25 clusters, DC-Clust and MS-MGKM were 67 and 47 times slower than Big-Clust, even on these datasets with fewer than 100,000 data points. Furthermore, their computational times increase exponentially with the number of clusters, as shown in Fig. 3. Consequently, we exclude these two methods from further comparisons.

In addition, DRS-Means usually performed well on moderate-sized datasets, although 1000 iterations were not always sufficient for an

[3] The f_{best} values were not preliminary known for the Range Queries Aggregates, CORD-19 Embeddings, and BitcoinHeist datasets.

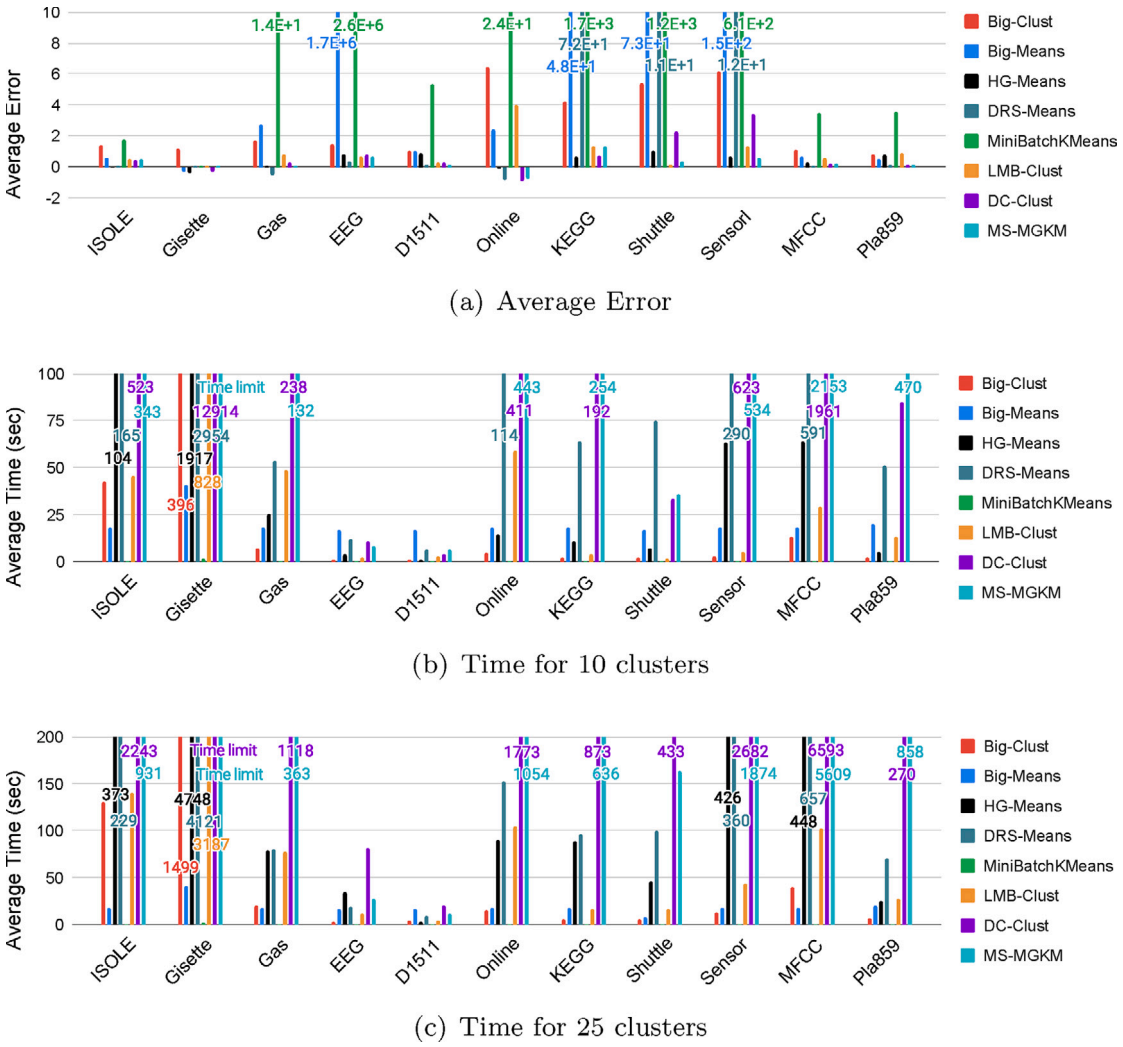


Fig. 1. Summary of results in datasets with less that 100,000 samples. Note: DC-Clust (MS-MGKM) exceeded the maximum time limit when attempting to form more than 15 (5) clusters in the Gisette dataset. The computational times for MiniBatchKMeans were extremely short and, therefore, not visible on this scale.

accurate solution (see Fig. 1(a)). However, 1000 iterations already impose a significant computational burden, thus prohibiting the use of a larger number of iterations. For datasets with fewer than 100,000 data points, DRS-Means was about four times slower than Big-Clust (average computational times for 25 clusters, see Fig. 1(c)), but for datasets with 100,000–500,000 data points, it was already 30 times slower than Big-Clust and 150 times slower than Big-Clust+ (see Fig. 2(c)), and it failed due to memory issues caused by the Scikit-learn function `NearestNeighbors` on datasets with more than 500,000 data points. Moreover, unlike all incremental algorithms, DRS-Means remained computationally expensive, requiring nearly the same long runtime even with fewer clusters (see Figs. 3 and 4). Consequently, we also exclude DRS-Means from further comparisons.

HG-Means produces solutions of exceptional quality, with an average Aver. $E_{\text{aver}} = -0.66\%$, indicating that its solutions are, on average, better than the best-known solutions in all datasets. However, HG-Means was also the most time-consuming method tested: on datasets with fewer than 100,000 data points, it was four times slower than Big-Clust (average computational times for 25 clusters, see Fig. 1(c)), and on datasets with more than 100,000 data points, it was over ten times slower than Big-Clust and 139 times slower than Big-Clust+ (see Fig. 2(c)). We exclude the HEPPMASS and CORD-19 Embeddings datasets from this comparison, as HG-Means reached the maximum time limit of 10 h when attempting to form 20 and 10 clusters, respectively. The computational time of HG-Means increased with the number of clusters, despite it is not an incremental algorithm producing solutions for smaller k (see, Figs. 3 and 4). Hence, we conclude that while HG-Means performs very well for smaller datasets, it is not well-suited for big data clustering.

For the remaining algorithms, LMB-Clust is the most accurate method tested (see Figs. 1(a) and 2(a)). It produced cluster structures with the lowest errors in 14 datasets out of 22 with an average of Aver. $E_{\text{aver}} = 1.20\%$. However, especially in big datasets, it was also the most time-consuming method (excluding HG-Means), and it failed to solve the clustering problem in two datasets, Cord-19 Embeddings and HEPMASS, due to memory issues. The proposed Big-Clust algorithm was approximately twice as efficient as LMB-Clust in datasets with fewer than 100,000 data points, and nearly four times faster in datasets with more than 100,000 data points. Moreover, Big-Clust+ was about 44 times faster than LMB-Clust in these larger datasets. The difference in computational times between Big-Clust and LMB-Clust depends on the ratio of the batch size m_B to the total number of data points m. When $m > 50,000$, m_B is always 2% of m, while for datasets with fewer data points, the percentage is higher — for instance, 13% in the ISOLET dataset. Due to the sampling process, Big-Clust requires some additional work compared to its successor, LMB-Clust, and this extra effort becomes more significant as the ratio of batch size to the total number of data points increases. Hence, the improvement in computational time of Big-Clust compared to LMB-Clust is smaller in datasets with fewer data points than in larger datasets. In addition, as

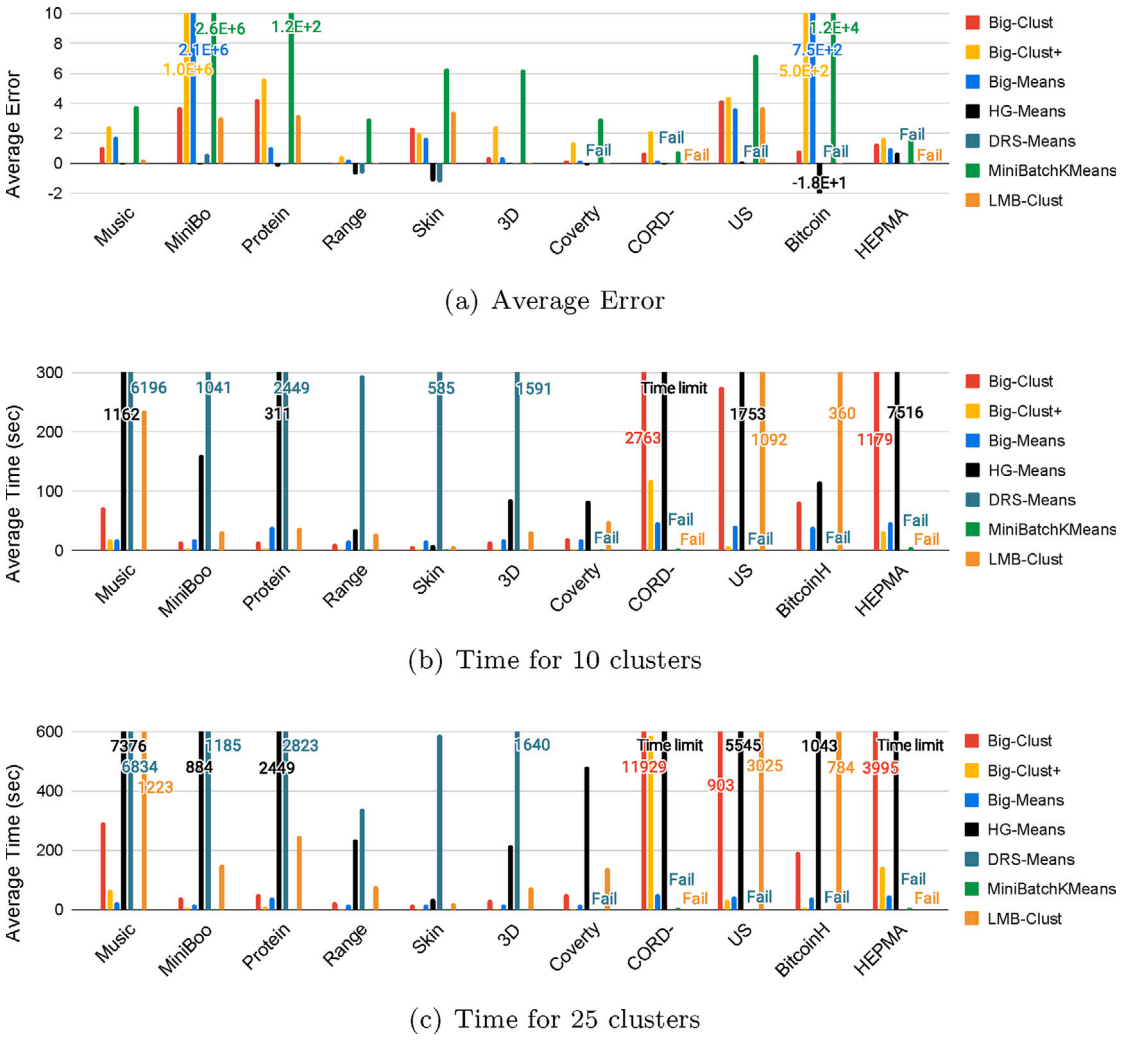


Fig. 2. Summary of results for datasets with more than 100,000 samples. Note: DRS-Means failed on all datasets exceeding 500,000 points, and LMB-Clust failed on the HEPMASS and CORD-19 Embeddings datasets due to memory issues. Additionally, HG-Means exceeded the maximum time limit when attempting to form more than 15 clusters in the HEPMASS dataset and more than 5 clusters in the CORD-19 Embeddings dataset. The computational times for MiniBatchKMeans were extremely short and, therefore, not visible on this scale.

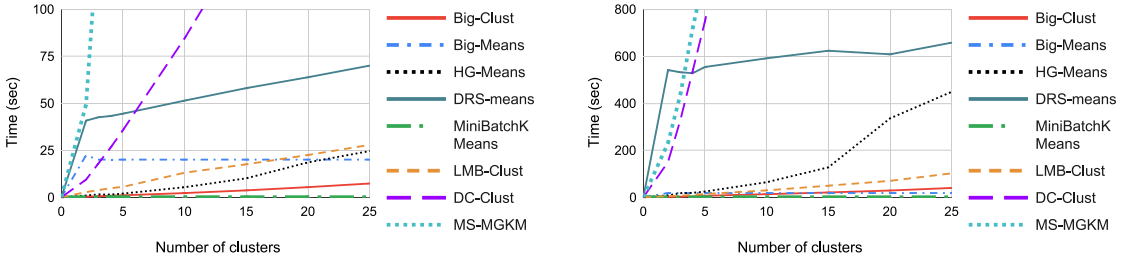
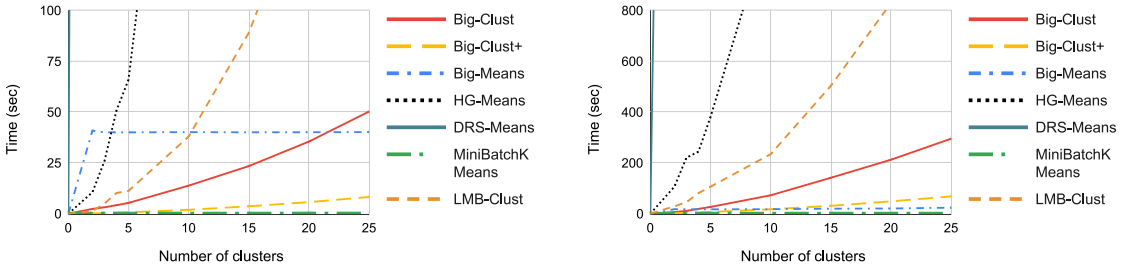


Fig. 3. Computational time versus number of clusters in datasets with different numbers of features: in Pla85900 $m = 85900$ and $n = 2$, while in MFCC $m = 85134$ and $n = 58$.

shown in Fig. 4, the computational time for Big-Clust, and especially for Big-Clust+, increases more slowly than for LMB-Clust as the size of the optimization problem (2) grows (recall that the number of variables in problem (2) is $n \times k$, where n is the number of features and k is the number of clusters). This is because the advantage gained from processing m data points in batches becomes more significant as the problem size increases. The average errors of Big-Clust are usually slightly larger (with an average of Aver. $E_{\text{aver}} = 2.26\%$) than those of the LMB-Clust except for Pla85900 and Skin Segmentation datasets, where Big-Clust produced cluster structures with lower errors on average.

The MiniBatchKMeans algorithm was clearly the most efficient method, taking at most a few seconds on all datasets (excluding t_{init}).



(a) Protein Homology (b) Music Analysis

Fig. 4. Computational time versus number of clusters in datasets with different numbers of features: in Protein Homology $m = 145751$ and $n = 74$, while in Music Analysis $m = 106574$ and $n = 518$. DRS-Means , which scales beyond these figures, required 2280 (5845) seconds for $k = 2$ and 2823 (6833) seconds for $k = 25$ on the Protein Homology (Music Analysis) dataset.

However, it completely failed (Aver. $E_{\text{aver}} > 10\%$) to find solutions close to the best-known solution in nine datasets. In almost all of these cases, $E_{\min}$ was also high, indicating that none of the ten executions of this algorithm were successful. If we omit these failures the average Aver. E_{aver} is still 3.46% for MiniBatchKMeans, which remains the highest error level observed in our experiments.

Additionally, Big-Clust+ failed completely (Aver. $E_{\text{aver}} > 10\%$) in two datasets, while Big-Means failed in six datasets. Nevertheless, with Big-Clust+, the $E_{\min}$ values are often low, meaning that at least one execution of the algorithm succeeded in these cases. The average Aver. E_{aver} without failures was 2.50% for Big-Clust+ and 1.10% for Big-Means.

There were no complete failures with Big-Clust, as Aver. E_{aver} remained below 6.5% across all datasets. Notably, Big-Clust produced clustering with a smaller average objective function value than MiniBatchKMeans in all datasets but Gisette (see Figs. 1(a) and 2(a)). Additionally, Big-Clust+ outperformed MiniBatchKMeans in terms of objective function value across all datasets except for HEPMASS. It is worth noting that the solutions for both Gisette and HEPMASS were close to the best-known solutions (Aver. $E_{\text{aver}} < 2\%$) for all methods.

Given these observations, we conclude that despite its efficiency, MiniBatchKMeans is not our top choice for big data clustering due to its low accuracy and frequent complete failures. While the quality of the solutions obtained with MiniBatchKMeans could potentially be improved by performing repeated runs and retaining the best result, it is worth noting that [25] reports that even 5000 independent runs of K-means or K-means++ do not consistently produce solutions of a quality comparable to those of HG-Means.

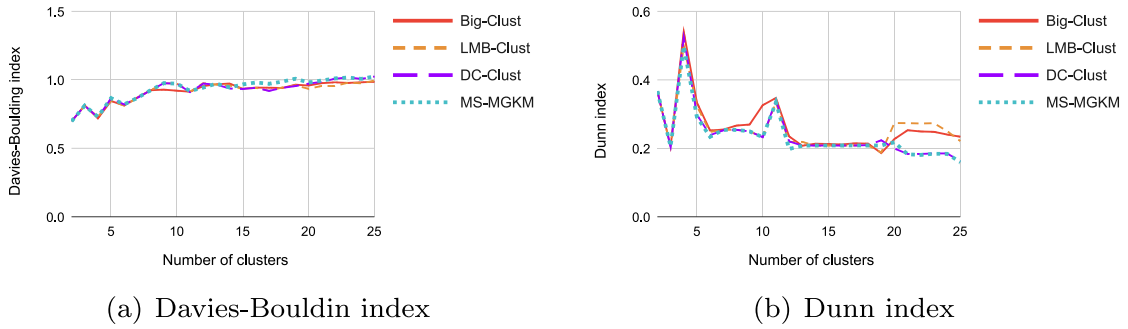
When comparing Big-Clust to Big-Means, we note that with $k = 10$ and $m < 500000$, Big-Clust was generally more efficient, except in the ISOLET, Gisette, and Music Analysis datasets (see Figs. 1(b) and 2(b)). These datasets contain a large number of features, which means that the number of variables in the optimization problem (2) is high. In addition, the ISOLET and Gisette datasets contain a relatively few data points, making the mini-batch approach used in Big-Clust less advantageous. For $k = 25$, Big-Means was faster also in the Gas Sensor Array Drift and MFCC datasets, as well as in all datasets with more than 100,000 data points (see Figs. 1(c) and 2(c)). This is because the computational time for Big-Means is predetermined and does not depend on the selected number of clusters (see Fig. 4). However, it is worth noting that Big-Clust also solves at the same time all intermediate problems for $l = 2, \ldots, 24$, while Big-Means only solves the problem for the given $k = 25$.

The errors produced by Big-Clust+ tend to be somewhat higher than those from Big-Clust and Big-Means, but it is the most efficient method in big data clustering (except MiniBatchKMeans, see Fig. 2). With $k = 10$, Big-Clust+ uses the least computational time in all but one dataset (Cord-19 Embeddings), and with $k = 25$, it loses to Big-Means only in the Cord-19 Embeddings, Music Analysis, and HEPMASS datasets. That is if we ignore the time t_{init} spent to read the data (see Tables S19 and S22). If we include it, Big-Clust+ is the most efficient method also in the Cord-19 Embeddings and HEPMASS. In fact, with t_{init} included, Big-Clust+ overrules also MiniBatchKMeans in these two datasets. We like to emphasize, though, that t_{init} is influenced by the programming language rather than the algorithm itself. In addition, while the runtime for Big-Means remains constant across different numbers of clusters ($t_k = t_{\max}$ + overheads), incremental algorithms like Big-Clust+ require more time as the number of clusters increases (see Fig. 4). Therefore, if a specific number of clusters, such as 75, is desired, Big-Means might be the better choice. However, Big-Clust+ provides all intermediate solutions, allowing users to select the best cluster structure and optimal number of clusters based on validity indices.

We note that some datasets are very sensitive to sampling, causing the failures mentioned above with Big-Clust+ and Big-Means. In particular, the results in EEG Eye State, KEGG Metabolic, Shuttle Control, and Sensorless Drive Diagnostic (see Fig. 1(a)) are extremely far away from the best-known solution with Big-Means that is not using whole data in computations. The same happens in MiniBooNE Particle Identification and BitcoinHeist with Big-Clust+ and Big-Means (see Fig. 2(a)), while with Big-Clust, the results are close to the best-known solution in all these datasets. This means that clustering in small subsets of data is more prone to all kinds of disruptions in data than clustering in entire data with the stochastic optimization method SLMBA.

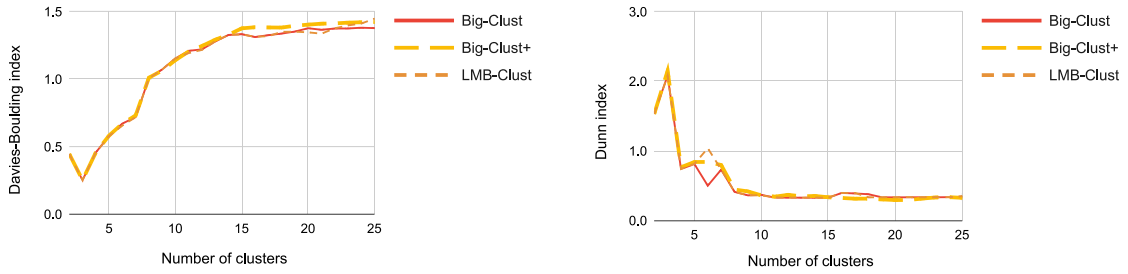
As stated in Remarks 1 and 3, the SLMBA (Algorithm 1) is a heuristic not converging to a stationary point of the clustering problem in general. Without results reported in tables, we mention that the average improvement in cluster function value is less than 1% in all the datasets, but KEGG Metabolic (1.24%) and Sensorless drive diagnostic (1.94%), if the globally convergent version of the SLMBA (Algorithm 2) is applied. When comparing the results with the LMB-Clust, the computational times usually improved slightly, but naturally, they are longer than with the heuristic SLMBA.

Fig. 5 demonstrates that the DBI and DI are similar with Big-Clust, LMB-Clust, DC-Clust, and MS-MGKM (see also Figures S1 – S11 in the supplementary material). In particular, based on these indices, all the algorithms identify the same optimal number of clusters for almost all datasets. Moreover, [13] reports that the LMB-Clust generally achieves similar or better index values — lower DBI and higher DI — compared to other incremental clustering algorithms tested. This indicates that the LMB-Clust (and thus also Big-Clust) is more effective at identifying well-separated clusters than the other algorithms. In addition, Big-Clust+ usually gives similar or comparable DBI and DI but with more disturbance (see, Fig. 6 and also Figures S11 – S22 in the supplementary material). This disturbance is a natural consequence as with Big-Clust+, the DBI and DI are computed only from part of the data. Thus, they do not necessarily provide accurate measures in whole datasets. Nevertheless, also in this case, the trends are the same, and optimal numbers of clusters based on the DBI and DI are similar to the other methods. It is worth noting that the results with the DBI are not necessarily consistent with those for the DI. In [13], it states that,



(a) Davies-Bouldin index (b) Dunn index

Fig. 5. Gas Sensor Array Drift: DBI and DI vs. number of clusters.



(a) Davies-Bouldin index (b) Dunn index

Fig. 6. US Census Data1990: DBI and DI vs. number of clusters.

in large datasets, the DBI is more accurate than the DI to identify the optimal number of clusters since the DI is very sensitive to noise. In practice, we could stop computing more clusters with an incremental algorithm, like Big-Clust, whenever the DBI starts to increase, thus accelerating the computation time further.

7. Conclusions

In this paper, we have proposed a doubly stochastic clustering algorithm Big-Clust for solving MSSC problems in large-scale and big datasets. Big-Clust is an incremental algorithm. Thus, in addition to the k-clustering problem with given k, it solves all the intermediate l-clustering problems with $l = 1, \ldots, k-1$. Further, we have introduced a stochastic limited memory bundle algorithm SLMBA for large-scale nonsmooth finite-sum optimization and used it as an underlying solver in the Big-Clust algorithm. It is worth noting that the SLMBA can also be applied to solve other finite-sum optimization problems common, for instance, in machine learning.

The new Big-Clust algorithm was tested on 22 large-scale real-world datasets (containing up to 10,500,000 data points) and compared with the state-of-the-art clustering methods Big-Means, HG-Means, DRS-Means, DC-Clust, MS-MGKM, MiniBatchKMeans, and LMB-Clust. The results show that the proposed Big-Clust algorithm is very efficient even in the biggest datasets, but it loses slightly in accuracy for the HG-Means and LMB-Clust. On the other hand, Big-Clust is not as sensitive to perturbation in data as Big-Means and MiniBatchKMeans, and it outperforms DRS-Means, DC-Clust, and MS-MGKM on big data.

We conclude that while DRS-Means, DC-Clust, and MS-MGKM are robust and efficient clustering algorithms in moderate-sized datasets, they are not well-suited for clustering in very large or big data due to their computational burden and memory requirements. If the goal is to achieve highly accurate clustering, HG-Means is the best option, despite being the most time-consuming algorithm tested (excluding DRS-Means, DC-Clust, and MS-MGKM). Conversely, MiniBatchKMeans was found to be too inaccurate, with arbitrarily high error values, to be considered a reliable clustering method, even though it was exceptionally fast. Therefore, due to its accuracy and efficiency, LMB-Clust is a recommended choice for clustering in datasets with less than 100,000 data points and 100 features. On the other hand, if the optimal number of clusters k is large (e.g., greater than 50) and known in advance, Big-Means is the preferred choice, as it avoids solving intermediate problems for $l < k$, making it more efficient in such cases. In all other scenarios, Big-Clust (or its variant Big-Clust+) is likely the best option, as its stochastic nature and incremental structure not only enable efficient clustering but also assist in determining the optimal number of clusters.

CRediT authorship contribution statement

Napsu Karmitsa: Writing – review & editing, Writing – original draft, Software, Methodology, Investigation. **Ville-Pekka Eronen:** Writing – review & editing, Investigation, Formal analysis. **Marko M. Mäkelä:** Writing – review & editing, Methodology, Investigation. **Tapio Pahikkala:** Writing – review & editing, Investigation, Funding acquisition. **Antti Airola:** Writing – review & editing, Methodology, Funding acquisition.

Funding

The work was financially supported by the Research Council of Finland, Projects No. #345804 and #345805 led by Tapio Pahikkala and Antti Airola.

Declaration of competing interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no financial support for this work that could have influenced its outcome.

Acknowledgment

We would like to thank Prof. Adil Bagirov for kindly providing the source code of MS-MGKM for testing purposes.

Algorithm 5: SLMBA

Input: A starting point $\boldsymbol{x}^1 \in \mathbb{R}^N$, the final accuracy tolerance $\varepsilon > 0$, the batch size $m_{\mathcal{B}} \leq M$, the termination limit $i_{term} > 1$, the iteration limit $i_{\max} > 0$, descent parameters $\varepsilon_L \in (0, 1/2)$ and $\varepsilon_R \in (\varepsilon_L, 1/2)$, the upper bound $t_{\max} > 1$ for serious steps, the distance measure parameter $\gamma > 0$, the number of stored correction vectors $m_c \geq 3$, and a finite-sum function f to be minimized.

Output: A heuristic minimizer $\boldsymbol{x}_{best} \in \mathbb{R}^N$ of function f.

Step 0. *Initialization:* Set $\boldsymbol{y}^1 \leftarrow \boldsymbol{x}^1$, $\beta^1 \leftarrow 0$, $f_{best} \leftarrow \infty$, and $h \leftarrow 1$.

Step 1. *Updating the best solution:* Compute $f(\boldsymbol{x}^h)$. If $f(\boldsymbol{x}^h) < f_{best}$, set

$$f_{best} \leftarrow f(\boldsymbol{x}^h), \qquad \boldsymbol{x}_{best} \leftarrow \boldsymbol{x}^h, \qquad \textit{and} \qquad i_{best} \leftarrow 0.$$

Else, set $i_{best} \leftarrow i_{best} + 1$.

Step 2. *Stopping criterion:* If $i_{best} > i_{term}$ STOP the algorithm with $\boldsymbol{x}_{best}$ as a best solution.

Step 3. *Selection of the batch:* Randomly select an index set $\mathcal{B}_h \subseteq \{1, \ldots, M\}$, with $|\mathcal{B}_h| = m_{\mathcal{B}}$. Compute $f_{\mathcal{B}_h}(\boldsymbol{x}^h)$ and $\boldsymbol{\xi}^h_{\mathcal{B}_h} \in \partial f_{\mathcal{B}_h}(\boldsymbol{x}^h)$.

Step 4. *Serious step initialization:* Set $\tilde{\boldsymbol{\xi}}^h \leftarrow \boldsymbol{\xi}^h_{\mathcal{B}_h}$ and $\tilde{\beta}^h \leftarrow 0$. Set $\hat{h} \leftarrow h$.

Step 5. *Direction finding:* If $h = 1$, set $\boldsymbol{d}^1 \leftarrow -\boldsymbol{\xi}^1_{\mathcal{B}_1}$ and go to Step 6. Else, compute

$$\boldsymbol{d}^h \leftarrow -D^h \tilde{\boldsymbol{\xi}}^h,$$

where D^h is computed by the L-BFGS update if $\hat{h} = h$ (use at most m_c correction vectors in S^h and U^h) and by the L-SR1 update, otherwise.

Step 6. *Stopping criterion for the batch:* Compute

$$w^h \leftarrow -(\tilde{\boldsymbol{\xi}}^h)^\top \boldsymbol{d}^h + 2\tilde{\beta}^h.$$

If $w^h < \varepsilon$, then $\boldsymbol{x}^h$ is the solution for the current batch: set $\boldsymbol{x}^{h+1} \leftarrow \boldsymbol{x}^h$, $h \leftarrow h + 1$, and go to Step 1.

Step 7. *Line search and auxiliary step:* Determine the step size $t^h_R \in (0, t_{\max}]$. Evaluate

$$\boldsymbol{y}^{h+1} \leftarrow \boldsymbol{x}^h + t^h_R \boldsymbol{d}^h \qquad \text{and} \qquad \boldsymbol{\xi}^{h+1}_{\mathcal{B}_h} \in \partial f_{\mathcal{B}_h}(\boldsymbol{y}^{h+1}).$$

Set $\boldsymbol{s}^h \leftarrow \boldsymbol{y}^{h+1} - \boldsymbol{x}^h = t^h_R \boldsymbol{d}^h$ and $\boldsymbol{u}^h \leftarrow \boldsymbol{\xi}^{h+1}_{\mathcal{B}_h} - \boldsymbol{\xi}^{\hat{h}}_{\mathcal{B}_h}$ (recall that $\hat{h}$ is the index of the last serious step). Append these values to S^h and U^h, respectively.

Step 8. *Serious step:* If

$$f_{\mathcal{B}_h}(\boldsymbol{y}^{h+1}) - f_{\mathcal{B}_h}(\boldsymbol{x}^h) \leq -\varepsilon_L w^h,$$

set $\boldsymbol{x}^{h+1} \leftarrow \boldsymbol{y}^{h+1}$, $f_{\mathcal{B}_h}(\boldsymbol{x}^{h+1}) \leftarrow f_{\mathcal{B}_h}(\boldsymbol{y}^{h+1})$, $\beta^{h+1} \leftarrow 0$, $h \leftarrow h + 1$, and go to Step 10.

Step 9. *Null step and aggregation:* Calculate the (stochastic) locality measure

$$\beta^{h+1} \leftarrow \max\{|f_{\mathcal{B}_h}(\boldsymbol{x}^h) - f_{\mathcal{B}_h}(\boldsymbol{y}^{h+1}) + (\boldsymbol{s}^h)^\top \boldsymbol{\xi}^{h+1}_{\mathcal{B}_h}|, \gamma \|\boldsymbol{s}^h\|^2\}$$

and determine multipliers $\lambda^h_i \geq 0$ for all $i \in \{1, 2, 3\}$, $\sum_{i=1}^3 \lambda^h_i = 1$ that minimize the function

$$\varphi(\lambda_1, \lambda_2, \lambda_3) = (\lambda_1 \boldsymbol{\xi}^{\hat{h}}_{\mathcal{B}_h} + \lambda_2 \boldsymbol{\xi}^{h+1}_{\mathcal{B}_h} + \lambda_3 \tilde{\boldsymbol{\xi}}^h)^\top D^h (\lambda_1 \boldsymbol{\xi}^{\hat{h}}_{\mathcal{B}_h} + \lambda_2 \boldsymbol{\xi}^{h+1}_{\mathcal{B}_h} + \lambda_3 \tilde{\boldsymbol{\xi}}^h) + 2(\lambda_2 \beta^{h+1} + \lambda_3 \tilde{\beta}^h),$$

where D^h is calculated by the same updating formula as in Step 5. Set

$$\tilde{\boldsymbol{\xi}}^{h+1} \leftarrow \lambda^h_1 \boldsymbol{\xi}^{\hat{h}}_{\mathcal{B}_h} + \lambda^h_2 \boldsymbol{\xi}^{h+1}_{\mathcal{B}_h} + \lambda^h_3 \tilde{\boldsymbol{\xi}}^h \qquad \text{and} \qquad \tilde{\beta}^{h+1} \leftarrow \lambda^h_2 \beta^{h+1} + \lambda^h_3 \tilde{\beta}^h.$$

Set $\boldsymbol{x}^{h+1} \leftarrow \boldsymbol{x}^h$, $h \leftarrow h + 1$ and go to Step 5.

Step 10. *Batch updating criterion:* If $\mod(h, i_{\max}) = 0$, go to Step 1. Else, set $\mathcal{B}_h \leftarrow \mathcal{B}_{h-1}$ and go to Step 4.

Appendix A

The more detailed SLMBA is given here as Algorithm 5. In numerical experiments, the default parameters similar to the LMBM [38] and LMB-Clust [13] are used.

Remark 9. The search direction d^h (Step 5 in Algorithm 5) is not necessarily a descending one — a common characteristic in NSO. Null steps (Step 9 in Algorithm 5) give further information on the nonsmooth objective whenever the search direction is not 'good enough'. On the other hand, the simple aggregation of stochastic subgradients would guarantee the global convergence of the method in the current batch if enough iterations in one batch were used.

Remark 10. If Algorithm 5 is used with $m_{\mathcal{B}} = M$, it reverts to the original LMBM with few extra iterations (loops from Step 1 to Step 6) before termination. Therefore, with $m_{\mathcal{B}} = M$ Algorithm 5 is globally convergent for LLC functions [38].

Appendix B. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.patcog.2025.111654.

Data availability

An open-source implementation of Big-Clust is available at https://github.com/napsu/bigClust The data used in experiments is public and the links are provided in the paper.

References

[1] R. Mussabayev, N. Mladenovic, B. Jarboui, R. Mussabayev, How to use k-means for big data clustering? Pattern Recognit. 137 (2023) 109269.

[2] A. Abdo, O. Abdelkader, L. Abdel-Hamid, SA-PSO-GK++: A new hybrid clustering approach for analyzing medical data, IEEE Access (2024).

[3] M.R. Karim, O. Beyan, A. Zappa, I. Costa, D. Rebholz-Schuhmann, M. Cochez, S. Decker, Deep learning-based clustering approaches for bioinformatics, Brief. Bioinform. 22 (1) (2021) 393–415.

[4] J. Sanjak, J. Binder, A. Yadaw, Q. Zhu, E. Mathé, Clustering rare diseases within an ontology-enriched knowledge graph, J. Am. Med. Inform. Assoc. 31 (1) (2024) 154–164.
[5] E. Riddle-Workman, M. Evangelou, N. Adams, Multi-type relational clustering for enterprise cyber-security networks, Pattern Recognit. Lett. 149 (2021) 172–178.
[6] S. Taheri, A. Bagirov, I. Gondal, S. Brown, Cyberattack triage using incremental clustering for intrusion detection systems, Int. J. Inf. Secur. 19 (2020) 597–607.
[7] L. Valtonen, S. Mäkinen, J. Kirjavainen, Advancing reproducibility and accountability of unsupervised machine learning in text mining: Importance of transparency in reporting preprocessing and algorithm selection, Organ. Res. Methods 27 (1) (2024) 88–113.
[8] W. Kim, A. Kanezaki, M. Tanaka, Unsupervised learning of image segmentation based on differentiable feature clustering, IEEE Trans. Image Process. 29 (2020) 8055–8068.
[9] T. Reutterer, D. Dan, Cluster analysis in marketing research, in: Handbook of Market Research, Springer, 2021, pp. 221–249.
[10] A. Bagirov, R. Aliguliyev, N. Sultanova, Finding compact and well-separated clusters: Clustering using silhouette coefficients, Pattern Recognit. 135 (2023) 109144.
[11] A. Bagirov, N. Karmitsa, S. Taheri, Partitional Clustering Via Nonsmooth Optimization: Clustering Via Optimization, second ed., Springer, Cham, 2025.
[12] A. Bagirov, J. Yearwood, A new nonsmooth optimization algorithm for minimum sum-of-squares clustering problems, European J. Oper. Res. 170 (2) (2006) 578–596.
[13] N. Karmitsa, A. Bagirov, S. Taheri, Clustering in large data sets with the limited memory bundle method, Pattern Recognit. 83 (2018) 245–259.
[14] A. Bagirov, S. Taheri, J. Ugon, Nonsmooth DC programming approach to the minimum sum-of-squares clustering problems, Pattern Recognit. 53 (2016) 12–24.
[15] Hoai An Le Thi, Minh Le Hoai, Tao Pham Dinh, New and efficient DCA based algorithms for minimum sum-of-squares clustering, Pattern Recognit. 47 (1) (2014) 388–401.
[16] N. Karmitsa, A. Bagirov, S. Taheri, New diagonal bundle method for clustering problems in large data sets, European J. Oper. Res. 263 (2) (2017) 367–379.
[17] W. Khalaf, A. Astorino, P. D'Alessandro, M. Gaudioso, A DC optimization-based clustering technique for edge detection, Optim. Lett. (2016) 1–14.
[18] A. Bagirov, B. Ordin, G. Ozturk, A. Xavier, An incremental clustering algorithm based on hyperbolic smoothing, Comput. Optim. Appl. 61 (1) (2015) 219–241.
[19] V. Xavier, A. Xavier, Accelerated hyperbolic smoothing method for solving the multisource fermat-weber and k-median problems, Knowl.-Based Syst. 191 (2020) 105226.
[20] S. Seifollahi, A. Bagirov, E. Borzeshi, M. Piccardi, A simulated annealing-based maximum-margin clustering algorithm, Comput. Intell. 35 (1) (2019) 23–41.
[21] S. Selim, K. Al-Sultan, A simulated annealing algorithm for the clustering, Pattern Recognit. 24 (10) (1991) 1003–1008.
[22] K. Al-Sultan, A tabu search approach to the clustering problem, Pattern Recognit. 28 (9) (1995) 1443–1451.
[23] Y. Alotaibi, A new meta-heuristics data clustering algorithm based on tabu search and adaptive search memory, Symmetry 14 (3) (2022) 623.
[24] T. Cura, A particle swarm optimization approach to clustering, Expert Syst. Appl. 39 (1) (2012) 1582–1588.
[25] D. Gribel, T. Vidal, HG-means: A scalable hybrid genetic algorithm for minimum sum-of-squares clustering, Pattern Recognit. 88 (2019) 569–583.
[26] P. Mansueto, F. Schoen, Memetic differential evolution methods for clustering problems, Pattern Recognit. 114 (2021) 107849.
[27] A. Bagirov, Modified global k-means algorithm for sum-of-squares clustering problems, Pattern Recognit. 41 (10) (2008) 3192–3199.
[28] A. David, S. Vassilvitskii, K-Means++: The advantages of careful seeding, in: SODA '07: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2007, pp. 1027–1035.
[29] H. Ismkhan, I-k-Means-+: An iterative clustering algorithm based on an enhanced version of the k-means, Pattern Recognit. 79 (2018) 402–413.
[30] O. Kozbagarov, R. Mussabayev, Distributed random swap: An efficient algorithm for minimum sum-of-squares clustering, Inf. Sci. 681 (2024) 121204.
[31] Y. Ping, H. Li, B. Hao, C. Guo, B. Wang, Beyond k-means++: Towards better cluster exploration with geometrical information, Pattern Recognit. 146 (2024) 110036.
[32] Z. Volkovich, D. Toledano-Kitai, G.-W. Weber, Self-learning k-means clustering: A global optimization approach, J. Global Optim. 56 (2) (2013) 219–232.
[33] D. Aloise, A. Deshpande, P. Hansen, Np-hardness of euclidean sum-of-squares clustering, Mach. Learn. 75 (2009) 245–248.
[34] V. Piccialli, A. Sudoso, A. Wiegele, SOS-SDP: An exact solver for minimum sum-of-squares clustering, INFORMS J. Comput. 34 (4) (2022) 2144–2162.
[35] A. Bagirov, N. Karmitsa, M. Mäkelä, Introduction To Nonsmooth Optimization: Theory, Practice and Software, Springer, 2014.
[36] B. Ordin, A. Bagirov, A heuristic algorithm for solving the minimum sum-of-squares clustering problems, J. Global Optim. 61 (2) (2015) 341–361.
[37] M. Haarala, K. Miettinen, M. Mäkelä, New limited memory bundle method for large-scale nonsmooth optimization, Optim. Methods Softw. 19 (6) (2004) 673–692.
[38] N. Haarala, K. Miettinen, M. Mäkelä, Globally convergent limited memory bundle method for large-scale nonsmooth optimization, Math. Program. 109 (1) (2007) 181–205.
[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[40] F. Clarke, Optimization and Nonsmooth Analysis, Wiley-Interscience, New York, 1983.
[41] A. Bagirov, M. Gaudioso, N. Karmitsa, M. Mäkelä, S. Taheri, Numerical Nonsmooth Optimization: State of the Art Algorithms, Springer, Cham, 2020.
[42] R. Byrd, J. Nocedal, R. Schnabel, Representations of quasi-Newton matrices and their use in limited memory methods, Math. Program. 63 (1994) 129–156.
[43] K. Kiwiel, Methods of Descent for Nondifferentiable Optimization, in: Lecture Notes in Mathematics, vol. 1133, Springer-Verlag, Berlin, 1985.
[44] M. Celebi, H. Kingravi, P. Vela, A comparative study of efficient initialization methods for the k-means clustering algorithm, Expert Syst. Appl. 40 (1) (2013) 200–210.
[45] B. Bahmani, B. Moseley, A. Vattani, S. Vassilvitskii, Scalable k-means++, Proc. VLDB Endow. 5 (7) (2012) 622–633.
[46] E. Forgy, Cluster analysis of multivariate data: Efficiency versus interpretability of classifications, Biometrics 21 (1965) 768–769.
[47] D. Davies, D. Bouldin, A cluster separation measure, IEEE Trans. Pattern Anal. Mach. Intell. PAMI-1 (2) (1979) 224–227.
[48] J. Dunn, Well-separated clusters and optimal fuzzy partitions, J. Cybern. 4 (1) (1974) 95–104.

Napsu Karmitsa received her Ph.D. in Scientific Computing from the University of Jyväskylä (Finland) in 2004. Since 2011, she has been an Adjunct Professor in Applied Mathematics at the University of Turku (Finland). Currently, Dr. Karmitsa serves as a Senior Research Fellow in the Department of Computing at the University of Turku. Her research focuses on nonsmooth optimization and data analysis, with an emphasis on nonconvex, global, and large-scale optimization and applications in machine learning.

Ville-Pekka Eronen received his M.Sc. in Mathematics from the University of Turku in 2018. The manuscript was completed while he was working in the Department of Computing at the University of Turku.

Marko M. Mäkelä obtained his Ph.D. in 1990 from the University of Jyväskylä. He currently holds a full professorship in Applied Mathematics at the University of Turku, where he also serves as Vice Head of the Department of Mathematics and Statistics. His research interests include nonsmooth and multiobjective optimization.

Tapio Pahikkala is a Professor of Machine Learning in the Department of Computing at the University of Turku, Finland, where he also received his doctoral degree in 2008. His current research interests and work encompass the theory and algorithms of machine learning, data analysis, and artificial intelligence, as well as their applications across various fields.

Antti Airola received his M.Sc. in Software Engineering in 2006 and his D.Sc. in Information and Communication Technology in 2011, both from the University of Turku. He is currently a Professor in the Department of Computing at this same university. His research interests include machine learning and data analytics, with a particular focus on their applications in the biomedical domain.