Fodor *et al.*
EURASIP Journal on Advances in Signal Processing (2022) 2022:108
https://doi.org/10.1186/s13634-022-00942-8

EURASIP Journal on Advances in Signal Processing

RESEARCH Open Access

A parallel ADMM-based convex clustering method

Lidija Fodor*, Dušan Jakovetić, Danijela Boberić Krstićev and Srđan Škrbić

*Correspondence:
lidija.fodor@dmi.uns.ac.rs

Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Trg Dositeja Obradovića 4, Novi Sad 21000, Serbia

Abstract

Convex clustering has received recently an increased interest as a valuable method for unsupervised learning. Unlike conventional clustering methods such as k-means, its formulation corresponds to solving a convex optimization problem and hence, alleviates initialization and local minima problems. However, while several algorithms have been proposed to solve convex clustering formulations, including those based on the alternating direction method of multipliers (ADMM), there is currently a limited body of work on developing scalable *parallel and distributed* algorithms and solvers for convex clustering. In this paper, we develop a parallel, ADMM-based method, for a modified convex clustering sum-of-norms (SON) formulation for master–worker architectures, where the data to be clustered are partitioned across a number of worker nodes, and we provide its efficient, open-source implementation (available on Parallel ADMM-based convex clustering. https://github.com/lidijaf/Parallel-ADMM-based-convex-clustering. Accessed on 10 June 2022) for high-performance computing (HPC) cluster environments. Extensive numerical evaluations on real and synthetic data sets demonstrate a high degree of scalability and efficiency of the method, when compared with existing alternative solvers for convex clustering.

Keywords: Distributed optimization, ADMM, High-performance computing, Performance evaluation

1 Introduction

Clustering represents a very common and widely present unsupervised learning problem in different areas [1–4]. The k-means algorithm [5] is a well-known broadly used clustering method. In addition, a number of useful clustering methods have been proposed, such as K-means++ [6], k-medians [7], and Bregman clustering [8].

As a valuable alternative, convex clustering has been proposed, e.g., [4, 9–12], wherein the clustering problem is formulated as a convex optimization problem based on a sum-of-norms penalty. Convex clustering exhibits several favorable features when compared with conventional clustering methods. For example, it eliminates local minima-related issues present in standard methods like k-means, and, unlike k-means, it does not need a pre-defined number of clusters to be specified. However, current convex clustering approaches do not scale well with the number of data points to be clustered [4, 9–12]. We develop a novel, fully parallel convex clustering-like method, i.e., a method for a

Springer Open

© The Author(s) 2022. **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

modified SON formulation, based on ADMM [13] amenable to master–worker (server–client) computational infrastructures. The proposed method improves scalability over existing convex clustering solvers. In addition, when compared with existing (parallel/distributed) clustering methods, e.g., those based on k-means, our method exhibits comparable accuracy (e.g., in terms of the silhouette score or percentage of accurately clustered points when the ground truth for expected outcomes is known) and scalability, while exhibiting convex clustering benefits (e.g., no need for a pre-defined number of clusters) with respect to the alternatives. We provide an open source implementation (available on [14]) of the proposed approach in the COMPSs parallel programming framework [15], based on the Python programming language and run on a High-Performance Computing (HPC) computer cluster.

1.1 Related work

k-means clustering was originally presented in [16], while Ref. [5] introduces the algorithm itself (Lloyd's algorithm). While k-means has been widely used and exhibits several useful features, it is sensitive to initialization and may converge to a local minimum. The impact of different initialization methods on the algorithm behavior was empirically evaluated in [17]. There have been several works that develop improved initialization methods. In [6], a randomized seeding technique was added to the algorithm, in order to improve both its speed and accuracy. On the other hand, Ref. [18] proposes an algorithm for cluster centers initialization.

Several convex clustering formulations have been considered. In [19], an exemplar-based likelihood function was introduced, leading to a convex minimization problem for clustering. This represents an efficient algorithm with guaranteed convergence to the globally optimal solution, proven also by a set of experiments. Reference [20] formulates an unsupervised learning problem as one convex "master" problem that includes non-convex sub-problems which can be solved efficiently. Supervised convex clustering has been proposed in [21].

The wider use of convex clustering becomes evident in different settings. For example, Ref. [22] investigates the use of convex clustering approach instead of hierarchical clustering in certain scenarios. An interesting approach was introduced in [23], where the idea is to perform sparse convex clustering. This means performing the clustering simultaneously with feature selection, in order to enhance performance. The Sum of Norms (SON) clustering was proposed in [4, 11, 12], as a convex relaxation of k-means. In [9], a detailed explanation of the algorithm for SON clustering can be found, including the presentation of its connection to k-means. It is inspired by the group lasso approach [24]. While the original SON clustering formulation involves all-pairwise-differences across cluster candidates in the SON regularization, it has been shown beneficial to utilize weighted pairwise differences, including setting up many weights to zero [10, 23, 25–29]. The weighted and sparse SON regularizations have been shown to yield faster algorithms and good clustering accuracies [10, 25, 28, 29]. For example, Qian [25] proposes an approach based on weighted minimum spanning trees and k-means bipartite graphs and shows high clustering accuracies of such sparse SON regularization methods. The SON clustering exhibits good theoretical cluster recovery guarantees for

all-pairwise-differences SON models. Theoretical advances regarding the perfect recovery properties of the convex clustering model with uniformly weighted all-pairwise-differences regularization were proven in [30, 31]. For weighted and sparse SON models, theoretical recovery guarantees are limited. Reference [32] establishes sufficient conditions for the perfect recovery guarantee of a general weighted convex clustering model. However, the weights have to be nonzero for all data point pairs within the same cluster—information not known a priori.

Several numerical algorithms for solving SON clustering problems have been proposed. Reference [32] introduces a semi-smooth Newton-based augmented Lagrangian method, for large-scale convex clustering. Additionally, the Alternating Direction Method of Multipliers (ADMM) and the Alternating Minimization Algorithm (AMA) have been proposed to solve convex clustering problems [10]. In [26], a novel method for convex clustering, using semi-proximal ADMM was introduced. This method is suitable for high-dimensional data. It is based on the sparse group lasso penalty and includes a set of numerical experiments in MATLAB. An interesting approach is presented in [27], where a networked k-means algorithm is proposed. The algorithm deals with distributed data and a multi-agent approach. Although it contains a description of an illustrative numerical evaluation, this work does not conduct a thorough empirical study. In [28], the authors introduce a Scalable cOnvex cLustering AlgoRithm via Parallel Coordinate Descent Method (SOLAR-PCDM). They combine a parallelizable algorithm with a compression strategy. SOLAR-PCDM includes the development of a method called weighted convex clustering to recover the solution path by formulating a sequence of smaller equivalent optimization problems and the utilization of the Parallel Coordinate Descent Method (PCDM) to solve a specific convex clustering problem. Reference [29] introduces an efficient smoothing proximal gradient algorithm (Sproga) for convex clustering.

With respect to the existing literature, our main contribution is on developing a novel parallel and scalable solver for SON-type clustering. We adopt a sparse zero-one weights SON formulation and leverage it to develop an efficient parallel ADMM method. Unlike existing ADMM-based convex clustering methods that are *sequential* [26], our method is parallel and hence, well suited to scalable execution on HPC clusters. Extensive numerical evaluations show a high clustering accuracy and a high scalability of the proposed method on a number of real and synthetic data sets. Specifically, the achieved accuracy is comparable to alternative sequential k-means solvers, while scalability is significantly improved. It is worth noting that our sparse zero-one SON formulation does not guarantee perfect theoretical recovery guarantees. However, extensive numerical results demonstrate a high clustering accuracy of the proposed method. This is a typical scenario with other sparse clustering methods like, e.g., [25, 26].

The rest of the paper is organized as follows. The definition of the problem is described in Sect. 2, where Sect. 2.1 is dedicated to the explanation of some introductory prerequisites. Section 2.2 contains the definition of the ADMM-based convex clustering-like algorithm. The implementation and infrastructure are described in Sect. 2.3, where Sect. 2.3.1 contains the description of the input data and Sect. 2.3.2 provides some remarks about the stopping criterion. Section 3 is dedicated to the numerical

evaluations. Section 3.1 provides insights into the time consumption of different segments of the algorithm. The accuracy evaluation of the developed method is described in Sect. 3.2. In Sect. 3.3, the effects of choosing different reference points (see ahead Sect. 2.2) are considered, while in Sect. 3.4, we evaluate the effects of different ways of data partitioning. The scaling properties of the algorithm are described in Sect. 3.5. Section 3.6 provides a discussion on choosing the value of parameter γ. The comparison of parallel ADMM-based convex clustering to other clustering approaches is given in Sect. 3.7, specifically: Sect. 3.7.1 contains the comparison the AMA method, Sect. 3.7.2 with DBSCAN and Sect. 3.7.3 with the SSNAL method. Section 3.8 discusses some possible further implementation considerations. Finally, the conclusions are made in Sect. 4.

Notation. We denote by: $\mathbb{R}^d$ the d-dimensional real coordinate space; $\|.\|$ the Euclidean norm; $a^\top$ the transpose of vector a.

2 Methods

2.1 Preliminaries

SON clustering. Consider the problem of clustering a set of observations $\{a_j\}_{j=1}^N$, $a_j \in R^d$, where the number of clusters is not known in advance. The Sum Of Norms (SON) clustering is formulated as:

$$\min_x \sum_{j=1}^{N} ||a_j - x_j||^2 + \gamma \sum_{i<j} ||x_i - x_j||, \tag{1}$$

where $x = \left((x_1)^\top, (x_2)^\top, \ldots, (x_N)^\top\right)^\top \in \mathbb{R}^{Nd}$ is the optimization variable. Here, $x_i \in \mathbb{R}^d$ plays the role of the i-th cluster center candidate, $i = 1, \ldots, N$ and $\gamma > 0$ is a regularization parameter. The first sum corresponds to fidelity measure, while the second sum represents the regularization term. It enforces zeros for $||x_i - x_j||$ across a subset of pairs i, j and can be seen as a generalization of the fused Lasso penalty [33]. This means that, at the solution $x^* = ((x_1^*)\top, \ldots, (x_N^*)^\top)^\top$ of (1), there will be only a subset of $K, K < N$, mutually distinct vectors $x_1^*, \ldots, x_N^*$; these K distinct vectors, say $x_{i_1}^*, \ldots, x_{i_K}^*$, where $\{i_i, \ldots, i_K\} \subset \{1, \ldots, N\}$ are the cluster centers obtained through convex clustering. The cost function in (1) is strongly convex, and (1) has the unique solution x^*.

ADMM. The proposed parallel clustering method is based on ADMM. ADMM [13] is an iterative algorithm that solves the following type of problems:

$$\text{minimize } f(x) + g(z) \text{ s.t. } Ax + Bz = c, \tag{2}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^m \to \mathbb{R}$ are convex functions, i.e., ADMM assumes an objective function, that is separable to two components, $x \in \mathbb{R}^n$ and $z \in \mathbb{R}^m$. $A \in R^{p \times n}$, $B \in R^{p \times m}$ are real-valued matrices, where $c \in R^p$. The augmented Lagrangian $L_\rho : (\mathbb{R}^{m+n}) \times (\mathbb{R}^p) \to \mathbb{R}$ associated with (2) is:

$$L_\rho(x, y; \lambda) = f(x) + g(y) + \lambda^\top (Ax + By - c) + \frac{\rho}{2} ||Ax + By - c||^2, \tag{3}$$

where $\rho > 0$ is a penalty parameter, and λ is the dual variable. Then, the ADMM algorithm consists of the following iterations:

$$x^{k+1} = \text{argmin}_x L_\rho(x, y^k, \lambda^k), \tag{4}$$

$$y^{k+1} = \text{argmin}_y L_\rho(x^{k+1}, y, \lambda^k), \tag{5}$$

$$\lambda^{k+1} = \lambda^k + \rho(Ax^{k+1} + By^{k+1} - c). \tag{6}$$

Here, $k = 0, 1, \ldots$, is the iteration counter, $x^k \in \mathbb{R}^n$ and $y^k \in \mathbb{R}^m$ are the primal variables, and $\lambda^k \in \mathbb{R}^p$ is the dual variable. It is well known that (x^k, y^k) converges to a solution of (2) under mild conditions; see [13].

Regarding the stopping criterion, a common way to terminate the algorithm is to introduce threshold values ϵ^{pri} and ϵ^{dual}, as feasibility tolerances, for the primal and dual feasibility conditions:

$$||r^k|| = ||Ax^k + By^k - c|| \leq \epsilon^{\text{pri}}, \tag{7}$$

$$||s^k|| = ||\rho A^\top B(y^k - y^{k-1})|| \leq \epsilon^{\text{dual}}, \tag{8}$$

so that the algorithm terminates if both conditions (7) and (8) are satisfied.

2.2 Problem model and proposed parallel method

Assume we have N observations $\{a_j\}_{j=1}^N \in R^d$ as stated before. We assume a master–workers (nodes) computational and sharing model with $K-1$ worker nodes that store data, perform calculations, and communicate with the master. Our goal is to construct an algorithm that performs cluster assignment, i.e., a function that maps a set of input data points to a discrete set of cluster labels. Without loss of generality, we index the master as the first node, and the workers as nodes $2, \ldots, K$. Each node (including master) has a chunk of the input data a of size $N/K \times d$[1]. To facilitate presentation, we introduce a two index notation, where $a_{ij} \in \mathbb{R}^d$ represents the j-th data point available at node $i, i = 1, \ldots, K, j = 1, \ldots, \frac{N}{K}$. We introduce a modification of standard convex clustering in (1), as follows. Note that the second term in (1) involves the differences across all pairs $(i, j), i < j$, of "candidate" cluster centers. Here, we also start by letting $x_{ij} \in \mathbb{R}^d, i = 1, \ldots, N, j = 1, \ldots, \frac{N}{K}$, be a "candidate" cluster center that corresponds to the (i, j)-th data point. However, unlike (1), we do not penalize the differences across all pairs of candidate clusters. Instead, we assign to the master a "reference point" x_{11}. Similarly, we assign to each worker $i, i = 2, \ldots, K$, a local "reference point" x_{i1}. The selection of reference points can be made arbitrarily, i.e., instead of choosing the first point from the data chunk, we could choose an arbitrary point at each worker. These points are not forced to become centers, and they have no advantage over other points in the process of determining the final cluster centers; they only represent an element for algorithm construction. The effects of choosing different reference points for the same experimental setup are described in details in Sect. 3.3.

We replace the second sum in (1) with the following sum:

[1] For simplicity, assume that N is divisible by K; otherwise, node 1 can take $\lfloor \frac{N}{K} \rfloor + r$ data points, and the remaining nodes take $\lfloor \frac{N}{K} \rfloor$ data points, where r is the remainder when dividing N by K.

$$\sum_{i=1}^{K}\sum_{j=2}^{N/K}||x_{i1}-x_{ij}||+\sum_{i=2}^{K}||x_{11}-x_{i1}||. \tag{9}$$

In other words, within the data points at each node $i, i=1,\ldots,N$, we penalize the difference between the local reference point x_{i1} and the remaining data points $x_{ij}, j=2,\ldots,\frac{N}{K}$, at that node. This corresponds to the first sum in (9). In addition, regarding cross-node penalization, we penalize the differences between the reference point on master x_{11} and the local reference points $x_{i1}, i=2,\ldots,K$. This corresponds to the second sum in (9). Finally, accounting for the sum of squared distances between each point a_{ij} and each candidate cluster center x_{ij}, we arrive at the following formulation for the convex clustering-like method:

$$\underset{x_{ij}}{\text{minimize}}\sum_{i=1}^{K}\sum_{j=1}^{\frac{N}{K}}||a_{ij}-x_{ij}||^2+\gamma\sum_{i=1}^{K}\sum_{j=2}^{\frac{N}{K}}||x_{i1}-x_{ij}||+\gamma\sum_{i=2}^{K}||x_{11}-x_{i1}||, \tag{10}$$

where the minimization is with respect to the variables $x_{ij}\in\mathbb{R}^d, i=1,\ldots,K, j=1,\ldots,\frac{N}{K}$, and $\gamma>0$. Note that formulation (10) depends on K, i.e., different values of K yield different optimization problems. Intuitively, for a larger K, the sum of norms penalty in (10) corresponds to an increasingly sparse graph, and one may expect that this results in a lower cluster recovering accuracy. We show by extensive experiments that the overall proposed method exhibits high clustering accuracy when K increases. Note that problem (10) is not equivalent to the analog of (1) below:

$$\text{minimize}\sum_{i=1}^{K}\sum_{j=1}^{N/K}||a_{ij}-x_{ij}||^2+\gamma\sum||x_{ij}-x_{lm}||, \tag{11}$$

where the second sum includes the differences between each pair of variables x_{ij} and x_{lm}. However, extensive numerical results show that solving (10) yields effective clustering methods.

As an intermediate step of the overall clustering procedure, for a given number of workers K, we add the following formulation to the described problem. It is useful to associate with problem (10) a graph $G=(\mathcal{N},E)$, where $\mathcal{N}$ is the set of N nodes, each corresponding to a single variable $x_{ij}, i=1,\ldots,K, j=1,\ldots,\frac{N}{K}$, and E is the set of edges $(i,j)\sim(l,m)$, such that there is an edge between nodes (i,j) and (l,m) if the second sum in (10) involves the term $||x_{ij}-x_{lm}||$. Figure 1 illustrates graph G on an example with $K=4$ nodes and $\frac{N}{K}=4$ data points per node. In Fig. 1, x_{11} corresponds to the reference point on master; $x_{i1}, i>1$ corresponds to the reference point on node (worker) i; and $x_{ij}, i>1, j>1$ correspond to the remaining candidate clusters. Formulation (10) penalizes differences $||x_{ij}-x_{lm}||$ for those pairs of x_{ij} and x_{lm} for which an edge in G exists.

In view of the graph-based representation of (10), the original convex clustering in (1) is recovered when G is replaced with the full (complete) graph. Similarly, (10) may be seen as a weighted convex clustering [32], where unit weights are added for those pairs of x_{ij}'s and x_{lm}'s where $(i,j)\sim(l,m)$, and zero weights are added elsewhere.

Note that we do not assume beforehand any knowledge of the "structure" or "distribution" of data across different nodes. Also, the graph construction is independent of the

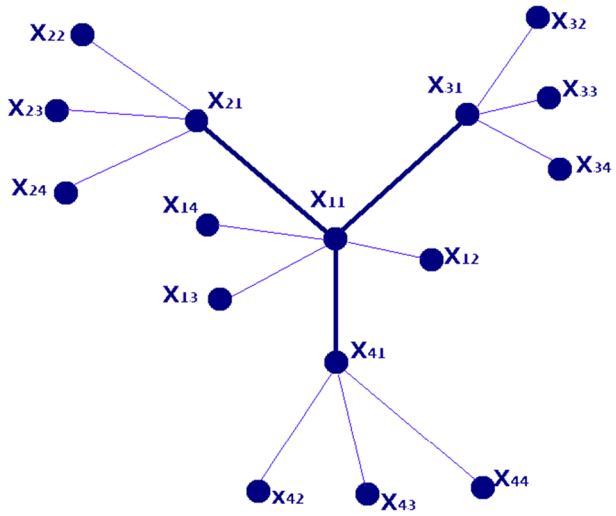

Fig. 1 Illustration of graph G and structure of problem (10)

actual values of the data points a_{ij}'s. In other words, the graph construction is arbitrary with respect to the available data. Extensive numerical results (See Sect. 3) show that this leads to accurate clustering solutions. We adapt this approach because the alternative, "data-driven" graph (weights) construction, as, e.g., done in [26], incurs high computational cost and communication coordination among nodes. Data-driven centers and graph assignments are left for future work.

The problem can now be reformulated, in order to apply ADMM, as follows:

$$\begin{aligned} \text{minimize} \sum_{i=1}^{K}\sum_{j=1}^{\frac{N}{K}} ||a_{ij}-x_{ij}||^2 + \gamma \sum_{i=1}^{K}\sum_{j=2}^{\frac{N}{K}} ||x_{i1}-x_{ij}|| + \gamma \sum_{i=2}^{K} ||x_{11}-y_{i1}|| \\ \text{s.t. } y_{i1} = x_{i1}, i = 2 \dots K. \end{aligned} \tag{12}$$

In other words, we introduce, for each node's reference point $x_{i1}, i = 1, \dots, K$, an auxiliary variable y_{i1} and add the constraint $y_{i1} = x_{i1}$ to keep problem (12) equivalent to (10). Clearly, variables in (12) are then $\{x_{ij}\}, i = 1, \dots, K, j = 1, \dots, \frac{N}{K}$, and $y_{i1}, i = 1, \dots, K$. We now dualize the constraints in (12) and form the Augmented Lagrangian function $L_\rho : \mathbb{R}^{Nd} \times \mathbb{R}^{Kd} \times \mathbb{R}^{Kd} \to \mathbb{R}$ as follows:

$$\begin{aligned} L_\rho(x, y; \lambda) = &\sum_{i=1}^{K}\sum_{j=1}^{\frac{N}{K}} ||a_{ij}-x_{ij}||^2 + \gamma \sum_{i=1}^{K}\sum_{j=2}^{\frac{N}{K}} ||x_{i1}-x_{ij}|| + \gamma \sum_{i=2}^{K} ||x_{11}-y_{i1}|| \\ &+ \sum_{i=2}^{K} \lambda_i^T (y_{i1} - x_{i1}) + \frac{\rho}{2} \sum_{i=2}^{K} ||y_{i1} - x_{i1}||^2, \end{aligned} \tag{13}$$

where $\rho > 0$ is a penalty parameter. In (13), we denote by $x \in \mathbb{R}^{Nd}$ the vector that stacks all the x_{ij}'s one in top of another, and by $y \in \mathbb{R}^{Kd}$ the vector that collects all y_{i1}'s one on top of another. We now apply ADMM in (4)–(6) with respect to the Lagrangian L_ρ in

(13) to solve (12). After decomposing x and y back to blocks x_{ij}'s and y_{i1}'s, it can be verified that (4)–(6) translates into the set of updates in (14)–(17), as follows:

- x update on each worker node $i = 2 \dots K$ in parallel:

$$x_{ij}^{k+1} = \text{argmin} \sum_{j=1}^{\frac{N}{K}} ||a_{ij} - x_{ij}||^2 + \gamma \sum_{j=2}^{\frac{N}{K}} ||x_{i1} - x_{ij}|| + (\lambda_i^k)^T (y_{i1}^k - x_{i1}) + \frac{1}{2}\rho ||y_{i1}^k - x_{i1}||^2 \tag{14}$$

  In (14), the optimization is (jointly) with respect to $x_{ij}, j = 1, \dots, \frac{N}{K}$.
- x update on the master node:

$$x_{1j}^{k+1} = \text{argmin} \sum_{j=1}^{\frac{N}{K}} ||a_{1j} - x_{1j}^k||^2 + \gamma \sum_{j=1}^{\frac{N}{K}} ||x_{1j} - x_{11}|| + \gamma \sum_{i=2}^{k} ||x_{11} - y_{i1}^k|| \tag{15}$$

  Note that (15) is also done in parallel with (14), and the optimization in (15) is (jointly) with regard to $x_{1j}, j = 1, \dots, \frac{N}{K}$.
- y update on master node:

$$y_{i1}^{k+1} = \text{argmin} \sum_{i=2}^{K} (\lambda_i^k)^T (y_{i1} - x_{i1}^{k+1}) + \frac{1}{2}\rho \sum_{i=2}^{k} ||y_{i1} - x_{i1}^{k+1}||^2 + \gamma \sum_{i=2}^{K} ||x_{11}^{k+1} - y_{i1}||. \tag{16}$$

  Note that minimization (16) is with respect to (jointly) variables $y_{i1}, i = 2, \dots, K$.
- λ update on master node:

$$\lambda_i^{k+1} = \lambda_i^k + \rho (y_{i1}^{k+1} - x_{i1}^{k+1}) \tag{17}$$

  Note that all the λ_i's, $i = 2, \dots, K$, are updated at the master independently, in parallel.

Regarding inter-node communications (variable exchange), the procedure is as follows. Assume for simplicity that all the λ's, x's and y's are initialized to zero. Then, after the master and the worker nodes update x according to (14) and (15), each worker i sends its new reference point variable x_{i1}^{k+1} to the master. After the master performs y and λ updates as in (16)–(17), it sends variables y_i^{k+1} and λ_i^{k+1} to worker $i, i = 2, \dots, K$.

We now comment on (14)–(17). At each iteration, each node and the master solve problems (14) and (15). These problems are of SON type, but with a sparse, star graph of SON penalties, and of variable size that is K times smaller than (10) and (11), hence, enabling scalability. Hence, for sufficiently large K, an efficient solver for moderate-sized SON problems can be adapted to solve (14) and (15), e.g., [29]. Actually, as detailed in Sect. 2.3, we used CVXPY as a general convex solver to solve (14)–(15). See also Sect. 2.3. Update (17) is clearly a cheap update. Finally, (16) is done closed form by evaluating a proximal operator block-thresholding for the 2-norm [26].

Note that formulation (10), unlike (11), does not guarantee perfect cluster recovery for any $\gamma > 0$. However, we observe numerically that, for the solution $\{x_{ij}^*\}$ of (10), an approximate clustering structure emerges. That is, the $\{x_{ij}^*\}$'s cluster into a number, say K', different groups, such that the x_{ij}^*'s within the same group are mutually very close. This motivates the following merging procedure.

Algorithm 1 Merging procedure of possible centers

On each node i locally, in parallel:
Require: $\epsilon_i, i = 1, ..., K; \epsilon$; initialize the list of local cluster centers $\mathcal{C}_i = \{\}$
  for all possible center candidates x_{ij}^* **do**
    for already accepted centers $c_{il} \in \mathcal{C}_i$ **do**
      if $\|x_{ij}^* - c_{il}\| \leq \epsilon_i$ **then**
        omit x_{ij}^* from centers $\mathcal{C}_i$
      else
        include x_{ij}^* to centers $\mathcal{C}_i$
      end if
    end for
  end for
  return the found local centers $\mathcal{C}_i = \{c_{i1}, ..., c_{iP_i}\}$, where P_i is the number of local centers found at node i
  On master node:
Require: ϵ;, All possible center candidates from the nodes: $\mathcal{C}_i, i = 1, ..., K$
Require: initialize the list of final centers $\mathcal{C} = Null$
  for all possible center candidates from all nodes $c_{ij} \in \mathcal{C}_i, i = 1, ..., K$ **do**
    for already accepted centers $c_l \in \mathcal{C}$ **do**
      if $\|c_{ij} - c_l\| \leq \epsilon$ **then**
        omit c_{ij} from centers $\mathcal{C}$
      else
        include c_{ij} to centers $\mathcal{C}$
      end if
    end for
  end for
  return the found centers $\mathcal{C} = c_1^*, ..., c_{P'}^*$, where P' is the final number of centers

Algorithm 1 shows the merging procedure, that is being applied after (14)–(17) converges. The first stage of merging is applied locally on each node. The threshold values ϵ_i and ϵ are positive numbers, used to filter the possible centers x_{ij}^*. We assign the first candidate point as a first center, and check the rest of the points. All those points that are close (within ϵ_i distance) to the points already marked as centers are being ignored. In the opposite case, a point is denoted as a new center. The second step is to merge the obtained local centers on the master node. This means that all the obtained local centers need to be synchronized on the master node first. The value ϵ is calculated by using the average distance between the obtained local centers. Then, the same merging procedure is applied as before. The result is a set of centroids on the output of the algorithm.

This means that we keep each next candidate for centroid only if its distance to previously kept centroids is larger than ϵ. Then, this whole set of already reduced subsets of possible centers from all nodes is being analyzed by the master node in the same manner. The output of this process is the set of resulting centers. The pseudocode for the described clustering algorithm is shown in Algorithm 2.

Algorithm 2 Pseudocode for the proposed algorithm

Require: global tuning parameters γ and $\epsilon^\star$, and at each node i: $a_{ij}, j = 1, ..., \frac{N}{K}$
  repeat
    Compute x_{1j}^k on master as in (15)
    Compute x_{ij}^k for each worker $i = 2..K$ in parallel as in (14)
    Compute y_{i1}^k on master as in (16)
    Compute λ_i^k on master as in (17)
  until a stopping criterion is met
  Merge the possible centers as described in Algorithm 1
Require: on each node i the final list of global centers $\mathcal{C} = c_1^\star, ..., c_{P'}{}^\star$
  for all local data points a_{ij} on each worker i, in parallel **do**
    assign point a_{ij} to cluster $c_l^\star$ where $min_{t=1,...,P'} \|a_{ij} - c_t^\star\| = \|a_{ij} - c_l^\star\|$
  end for

The overall proposed clustering method is summarized in Algorithm 2. After (14)–(17) converges and the merging procedure is applied as described in Algorithm 1, the master node makes the final centers available to the worker nodes. Then, each worker assigns its local data points to the cluster that corresponds to the nearest center. The algorithm works with 2 tunable parameters, γ and ϵ. Choosing a large value for the regularization parameter γ enforces more overlapping centroids. On the other hand, choosing a small value for the parameter can result in only slightly moving the centers, producing large distances between the gathered points, and hence, a too large number of clusters. See also Sect. 3.6.

One way to choose $\epsilon_i, i = 1, \ldots, K$ is the following:

$$\epsilon_i = \frac{2K^2}{\epsilon^\star \times N(N-K)} \sum_{j=1}^{\frac{N}{K}} \sum_{l=j}^{\frac{N}{K}} \|a_{ij} - a_{il}\|, \tag{18}$$

where $\epsilon^\star$ is a tunable input parameter that can also be set to a universal, data and problem independent, constant value, e.g., equal to 5 or 10. The formulation in (18) means that each node calculates the average Euclidean distance between the points in its own data chunk and divides the result by a constant value $\epsilon^\star$. Similarly, ϵ can be calculated on master, based on the average Euclidean distance between the locally obtained centers as follows:

$$\epsilon = \frac{2}{\epsilon^\star \times P(P-1)} \sum_{j=1}^{P} \sum_{l=j}^{P} \|c_j - c_l\|, \tag{19}$$

where $P = P_1 + \cdots + P_K$ is the overall number of locally obtained centers on all nodes and the sum in (19) involves all elements from the union of sets $\mathcal{C}_i, i = 1, \ldots, K$.

Calculating ϵ_i as in (18) involves only pairwise distances within single workers data, which is only $O(\frac{N^2}{K^2})$ pairwise distances. That is, across all nodes, we calculate $O(K \times \frac{N^2}{K^2})$ pairwise distances. Compared with $O(N^2)$ pairwise distances, needed with sequential weighted SON clustering approaches, like for example AMA [10], it is significantly cheaper for sufficiently large number of workers K. Alternatively, the sums in (18) may be replaced with minimum, i.e., to consider minimal within-workers data distances. When

performing the merging of the local centers, the value of ϵ on the master can be used by calculating it in the same manner as in (18), with the only difference that the distances of local centers are considered instead of the distances of data points.

Algorithm 1 may be seen as a simple instance of a pairwise clustering method; see, e.g., [34]. Intuitively, solving (10) usually brings the $x_{ij}^{\star}$'s that correspond to the data points a_{ij}'s within a single "true" cluster very close to each other, but it may not make them exactly equal up to the full accuracy. Therefore, a simple pairwise clustering method in Algorithm 1 is introduced to fine-tune the results achieved by solving (10). Note that Algorithm 1 involves $O(K \times \frac{N^2}{K^2})$ pairwise comparisons across all nodes, and hence, again it scales well when K is large. We also report that Algorithm 1 allows for a cheap polishing of the results, typically incurring 11.6% of the overall execution time of Algorithm 2, on average.

2.3 Implementation and infrastructure

The implementation of Algorithm 2 is developed in Python, using the COMPSs framework for parallel execution [15]. COMPSs offers a simple programming model with the aim to facilitate the parallelization process. It has been widely adopted and extended in numerous scientific projects offered as a tool to develop scientific applications and optimize their execution on distributed infrastructures. The testing has been performed on the AXIOM computing facility consisting of 16 nodes (8 x Intel i7 5820k 3.3GHz and 8 x Intel i7 8700 3.2GHz CPU-96 cores and 16GB DDR4 RAM/node) interconnected by a 10 Gbps network.

The Python implementation relies on the CVXPY [35, 36] package, used for the minimizations described in (14)–(15). The PyCOMPSs framework [37] (COMPSs for Python) enables a convenient way for parallelization, by simply annotating a function as a task. However, this requires a proper data format and distribution, as well as a synchronization point, where the results of execution on different processes are being collected into a predefined data structure.

2.3.1 The input data

The input data are read from a file and resized from its two-dimensional form to a three-dimensional form $workers \times chunk\_size \times d$, where *workers* is the number of nodes operating (defined as an input parameter) and $chunk\_size = \frac{N}{K}$. The data points are being distributed in consecutive chunks as read from files. This means that the data distribution across workers is not "informed" and is arbitrary, i.e., each worker usually contains a mixture of data points that should belong to different clusters. The tests are based on both synthetical and real data sets. The synthetical data sets were generated in order to test scalability and accuracy of the algorithm. The data sets are generated by using a samples generator from the scikit-learn package [38]. Figure 2 represents an example for generated two-dimensional data set of small volume. It contains 30 points, with clearly distinguishable clustering into 3 clusters.

Large synthetic data sets are generated as Gaussian mixture models [39]. We consider mixtures of k multivariate Gaussian distributions with mean μ_i and covariance $\mathcal{E}_i, i = 1 \ldots k$. The values for μ are d-dimensional points generated randomly, but from different intervals for each Gaussian k, in order to ensure that they will be distant

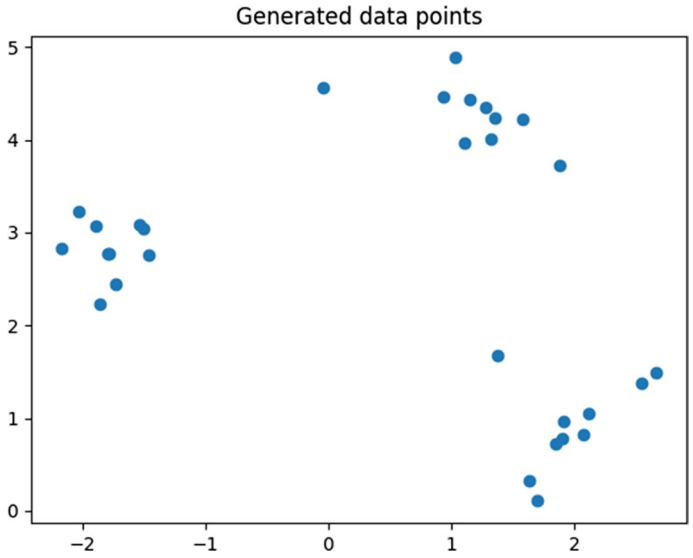

Fig. 2 Example of a two-dimensional data set of small volume

Fig. 3 t-SNE for an example of the generated three-dimensional data set of larger volume

enough to represent separate clusters. Also, the values for σ are generated randomly, from one interval for the diagonal part and from another one for the upper triangular part. Regarding the value of π, the same value of $\pi = \frac{1}{k}$ was used for each Gaussian. Figure 3 shows the t-SNE embedding [40] for an example of generated data. The data set size is 3000 points here, with dimension 3 and 5 clusters.

2.3.2 The stopping criterion

The stopping criterion implemented here is the usual stopping approach for ADMM. It requires implementing (7)–(8).

The residuals are being calculated at the end of each iteration, when the master process has access to all results, obtained by the workers. The threshold values ϵ^{pri} and ϵ^{dual} are also being recalculated at each iteration as follows:

$$\epsilon^{\text{pri}} = \alpha\sqrt{N} + \beta\max\{||x^k||, ||-y^k||\}, \tag{20}$$

$$\epsilon^{\text{dual}} = \alpha\sqrt{N} + \beta||\lambda^k||. \tag{21}$$

Here, $\alpha \in R$ and $\beta \in R$ represent the absolute and relative tolerance values, respectively. These values can be set as input parameters and their default values are $\alpha = 10^{-4}$ and $\beta = 10^{-2}$. This means that both the residuals and the threshold values are being updated at the end of each iteration k, based on the values x^k, y^k and λ^k.

3 Results and discussion

In this section, the aim is to assess the quality of different aspects of the distributed ADMM-based convex clustering-like algorithm. Particularly, the tests performed on small, two-dimensional data sets are appealing for plotting and evaluating the properties of the algorithm. One of the main ideas is to monitor the accuracy of the solution, which is straightforward with data sets generated under controlled conditions. For these synthetic data sets, the number of expected clusters is known in advance, which makes the first stage of the evaluation simpler. However, we will also include other accuracy metrics, as silhouette score and comparison with the results of plain k-means clustering. The second key aspect is the evaluation of performance. The scaling properties of the algorithm will be demonstrated through a set of tests on different data sets.

First, we discuss the time consumption of different segments of the algorithm in Sect. 3.1. Section 3.2 is dedicated to accuracy evaluation. These analyses include measuring the percentage of accurately clustered points, where a ground truth is known. We use a small, synthetic data set and the Iris data set [41, 42] here. Additionally, we observe the accuracy of some large, synthetic data sets, by means of silhouette score values and comparison with k-means. Section 3.3 contains an evaluation of the effects of choosing different reference points, while Sect. 3.4 considers the effects of different ways of input data partitioning. In Sect. 3.5, we evaluate the scalability of the proposed method, on some large, synthetic data sets, generated as Gaussian mixtures. Further, we discuss some aspects of choosing the value for the regularization parameter γ in Sect. 3.6. Section 3.7 is about comparing the proposed method to other clustering approaches: the AMA method (Sect. 3.7.1), DBSCAN (Sect. 3.7.2) and SSNAL (Sect. 3.7.3). We perform the comparison on synthetic data sets, generated as Gaussian mixtures. We conclude the numerical evaluations in Sect. 3.8, where some possibilities for further enhancement are mentioned.

3.1 Time consumption of different segments of the algorithm

When considering the time consumption of the different actions during the algorithm execution, it naturally emerges that the iterative part of algorithm consumes 88.3% of the overall execution time, on average. The average time spent on reading the input data is only 0.05%, while the average time needed for the process of merging the possible

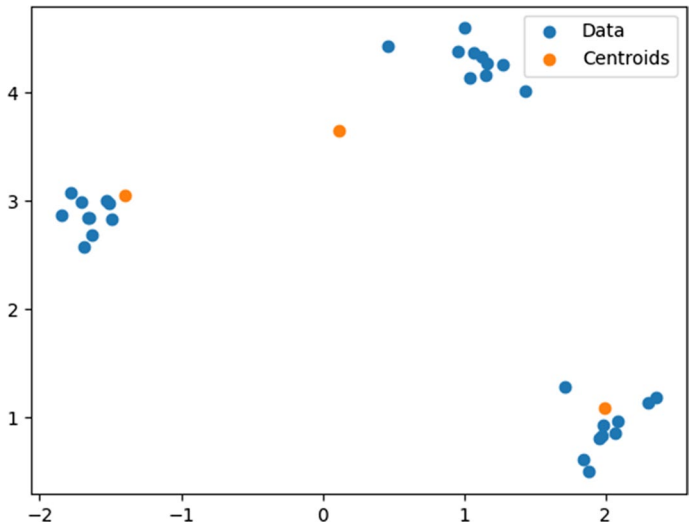

Fig. 4 Results of clustering for a generated data set $30 \times 2, \gamma = 0.3, \epsilon^{\star} = 2$

centers is 11.6%. These average values are calculated over all the experiments conducted and mentioned in the paper.

3.2 Accuracy evaluation

In order to gain some insights into the level of accuracy of the developed clustering implementation, a few approaches are used here. First, we investigate the solution for a small, generated two-dimensional data set, where it is straightforward to plot the results and gain visual insights. Figure 4 shows an example with 30 two-dimensional data points. The points belong to 3, clearly separable clusters, as shown by the blue dots in Fig. 4. The results of clustering are also shown in Fig. 4. Clearly, the algorithm is able to identify the centers correctly. Figure 4 displays the found centroids, as shown by the orange dots. These centers are the output from the merging mechanism. When the stopping criterion is met, the center candidate points, that are "close", by the definition of ϵ_i (18) are being aggregated. It can be seen that the algorithm properly detected the existence of 3 clusters and assigned the points to the clusters accurately.

The accuracy evaluation, when the ground truth is known, can be illustrated on a common example of the Iris data set. This means dealing with a higher-dimensional data set, where the clusters are mainly distinguishable by the nature of the underlying data.

When ground truth is available, we evaluate clustering accuracy as the percentage of correctly classified data points. The Iris data set [41, 42] is available in scikit-learn. It contains 3 different classes of the plant Iris. It has 4 attributes and 150 samples. Based on these attributes, a clustering algorithm could identify the existence of 3 clusters (see Table 1). We executed the ADMM-based convex clustering-like algorithm on this data set and obtained the 3 clusters. In order to further analyze the results, we compared the obtained labels to the real, known labels. It turns out that our algorithm assigned 93.33% of the data points to the correct cluster. It should be also mentioned that it assigns all data points belonging to the first cluster accurately, while it makes some 'mistakes' with the second and third cluster. The nature of the data directly affects this, as the mentioned

Table 1 Accuracy comparison for different clustering algorithms on the Iris data set

Algorithm	Parameters	Number of clusters	Accuracy (percentage)
ADMM-based convex clustering-like algorithm	$\gamma = 40, \epsilon^\star = 5$	3	93.33
k-means clustering	$k = 3$	3	88.66
AMA clustering	$\gamma \in [4.3, \ldots, 9.1]$	3	90.66

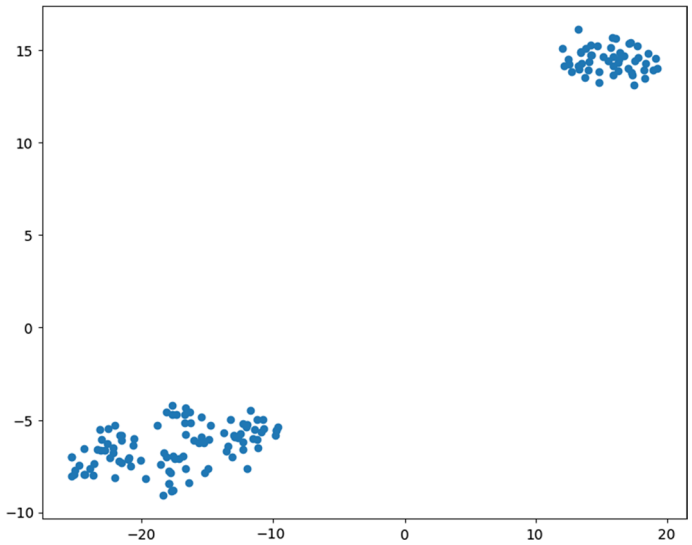

Fig. 5 The t-Sne embedding of the Iris data set

two clusters are 'close' to each other. In fact, running the standard k-means algorithm on this data set for different values of k results with a highest silhouette score value of 0.68 for $k = 2$. This can be also easily seen on the t-SNE embedding of the data set, in Fig. 5. Table 1 also contains the percentages of accurately clustered points for the standard k-means algorithm (with the preset value of k) and for the AMA method as well, for reference. When running plain k-means for $k = 3$, the percentage of points accurately clustered is 89.33%. As these evaluation showed, our method can perform as accurately as (or even more accurately than) k-means with a correctly predefined value for k. In order to further investigate this test case, we also run the AMA method [10] on this data set. By setting the parameter γ appropriately, it is able to find the 3 clusters with 90.66% of points clustered accurately. This shows that the 3 different clustering algorithms perform with a similar degree of accuracy on this data set. This is illustrated in Fig. 6. It shows the accuracy percentage related to the value of the input parameter γ. The accuracy percentage of k-means is independent regarding the value of the input parameter γ. Different initializations for k-means, meaning the usage of alternate algorithms as 'elkan' and 'full' or number of runs with different centroid seeds set to {5, 10, 100}, always result with the same level of accuracy of 89.33% on this data set. The AMA method has its range $\gamma \in [4.3, \ldots, 9.1]$, where it produces the highest accuracy. All values of γ that are out of this range affect significant decrease in clustering accuracy. The similar holds for ADMM-based convex clustering-like method, except that this range of values giving the

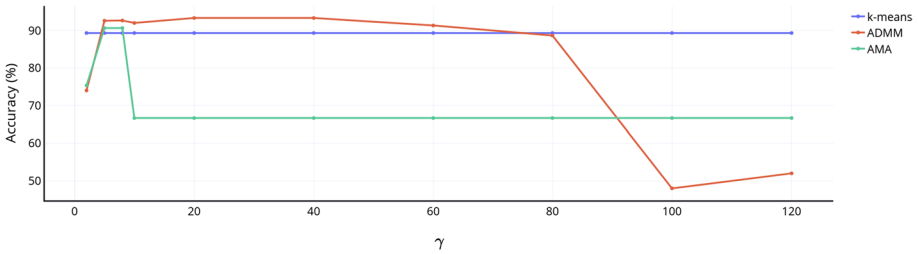

Fig. 6 The accuracy values of different methods on Iris data set

Table 2 Accuracy evaluation for higher dimensional data sets

Data size	γ	$\epsilon^\star$	Clusters ADMM-based convex clustering-like method	Clusters k-means	ADMM-based convex clustering-like method s.score	k-means s.score
1000×3	5.0	4	8	8	0.76	0.76
5000×3	6.6	2	4	4	0.77	0.78
5000×5	6.6	2	5	4	0.62	0.75
$10{,}000 \times 3$	6	5	10	10	0.69	0.75

highest accuracy is significantly broader. Intuitively, even outside of the range of γ for which a SON-like clustering is exact, in a vicinity of this range, "SON-like clustering still produces nearly-exact" clustering—that is then harnessed for correct clustering—via the merging procedure.

As the ground truth for clustering is not always available, some additional accuracy metrics can be used in order to assess the outcomes of clustering. Silhouette score is a common and widely used way to evaluate a clustering approach, so it is included as an accuracy metric in our experiments.

The accuracy of the algorithm is also tested for large higher-dimensional data sets. These data set are generated as Gaussian mixture models, as described earlier (see Sect. 2.3.1). In order to visualize the results, t-distributed stochastic neighbor embedding (t-SNE) [40] will be used. Let us consider a few higher-dimensional data sets. Table 2 shows the results for these experiments. It can be seen that the algorithm is mostly able to identify the expected number of clusters, with high silhouette score values. Additionally, the scikit-learn k-means algorithm results mostly with the same number of clusters and similar values for silhouette score.

Figure 7 shows the t-SNE embedding for a 1000×3 generated data set. The data points are colored according to their cluster labels, obtained by our ADMM-based convex clustering-like algorithm. It represents an example where our clustering algorithm clusters the data points accurately to the expected clusters, that also corresponds to a high silhouette score value.

Based on the described experiments, it can be concluded that ADMM-based convex clustering-like method can perform with high accuracy. The accuracy of the algorithm is compatible with the accuracy level of k-means and AMA.

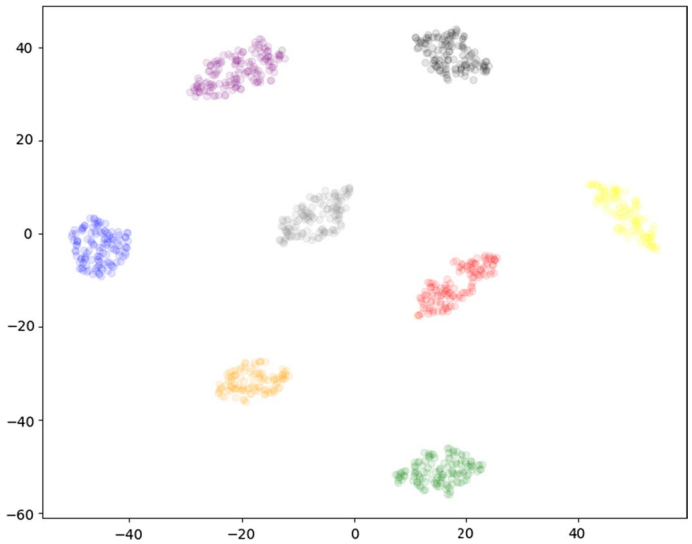

Fig. 7 t-SNE embedding for clustering over a synthetical data set of size 1000×3

Table 3 Accuracy for a set of tests with different selections of reference points, and fixed input parameter values

Data set	Average accuracy (%)	Minimal accuracy (%)	Maximal accuracy (%)	Median accuracy (%)	0.98 quantile (%)
120×2	100	100	100	100	100
1000×4	99.87	99.2	100	99.9	99.9
Iris	90.67	48.67	91.33	91.33	91.33
Iris-with optimized γ	90.98	84	91.33	91.33	91.33

3.3 The effects of choosing different reference points

The problem formulation in (10) proposes, for the sake of proper definition and without loss of generality, the first point (according to the adopted points enumeration) in each worker's local data chunk to become the local reference point, and the first point in the data set (which is also the first point on master's local data chunk) to become the global reference point. However, arbitrary points can be selected as reference points, without serious accuracy losses. Let us demonstrate this on a few examples. We consider 3 different data sets: a small synthetic data set with dimension 120×2 (generated using samples generator from the scikit-learn package [38], that generates isotropic Gaussian blobs for clustering), a larger synthetic data set with dimension 1000×4 (generated as a Gaussian mixture model) and the Iris data set. Note that we first reshuffled the points for the Iris data set here, as they are originally arranged in order, by clusters. If we use 3 workers, this means that each worker gets data points from one cluster, for the original ordering. Instead of this, we want data points from different clusters on one worker, so that when we pick a reference point position, it could correspond to a point from any cluster. For all these considered data sets, the ground truth is known, so the accuracy percentage can be easily obtained. Table 3 shows the average, minimal, maximal and median accuracy values and the 0.98 quantile for these data sets, obtained for randomly selected reference points, within 500 tests for each data set. The accuracy is defined, based on

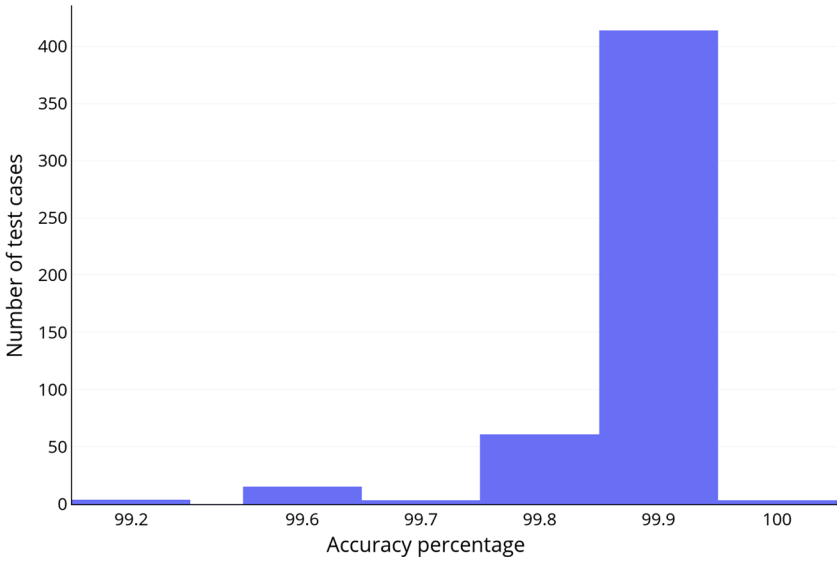

Fig. 8 Histogram for accuracy percentage for different selections of reference points, on a synthetic data set of size 1000×4

the ground truth here. When the algorithm determines the labels of the data points, we compare those labels with the real ones (considering that the label marks may be different now, for example, a cluster labeled with 1 originally can be labeled with 2 now). The random selection of reference points means that we allow for each worker to generate a random value for the reference point position, with the limitation that we never use the same reference points positions combination during the tests. It is important to mention that the input parameters γ and $\epsilon^\star$ are fixed for each data set during these tests. For the smallest data set with 120×2 data points, all the 500 test cases with different reference points resulted with 100% accuracy. For the larger data set with 1000×4 data points, the best accuracy obtained was 100%, and the lowest value was 99.2%, which certainly does not represent a significant difference. In fact, in the vast majority of tests, the accuracy is 99.9%. This can be seen in Fig. 8 that displays the histogram of the achieved accuracies across all 500 trials. The third test case, regarding the Iris data set, also shows a consistent accuracy level in most of the cases. However, there are a few exceptions here, as it can be seen in Table 3. The lowest accuracy obtained for the Iris is 48.67% that occurs in only 1.4% of the test cases. The test cases resulting with this value can be drastically improved by slightly changing the input parameter γ, so that they result with high accuracy values, similar as the rest of the tests. The values and amounts of accuracy percentages obtained for the Iris data set, with a fixed value for parameter γ, are also shown in Fig. 9. When we adjust the parameter γ, we can eliminate the low accuracy values. This is shown in the last row of Table 3 and also in Fig. 10. If the reference points are changed, one can adapt the parameter γ easily (e.g., by running a clusterpath), in order to maintain high accuracy. It is evident that the method is able to achieve a high accuracy level, for different selections of reference points, on different data sets.

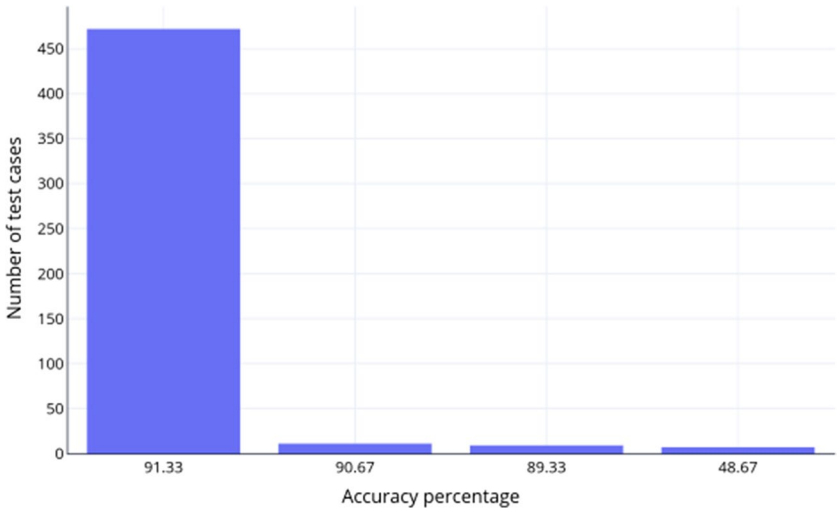

Fig. 9 Histogram for accuracy percentage for different selections of reference points, on the Iris data set

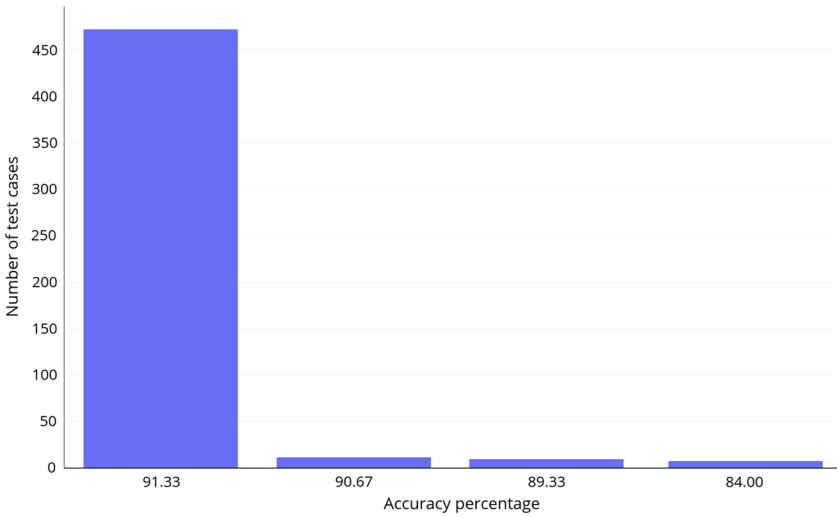

Fig. 10 Histogram for accuracy percentage for different selections of reference points, on the Iris data set, with optimized γ

Table 4 Accuracy for a set of tests for differently shuffled input data

Data set	Average accuracy (%)	Minimal accuracy (%)	Maximal accuracy (%)	Median accuracy (%)	0.95 quantile (%)
120 × 2	100	100	100	100	100
1000 × 4	99.1	80.8	100	88.66	100
Iris	89.22	86	93.33	90.66	91.6

3.4 The effects of different ways of data partitioning

The way a data set is partitioned among a set of workers is an aspect that may influence the outcome of the algorithm. Therefore, we evaluate this aspect, on 3 different data sets: the synthetic 120 × 2 and 1000 × 4 data sets and on the Iris data set again. Table 4

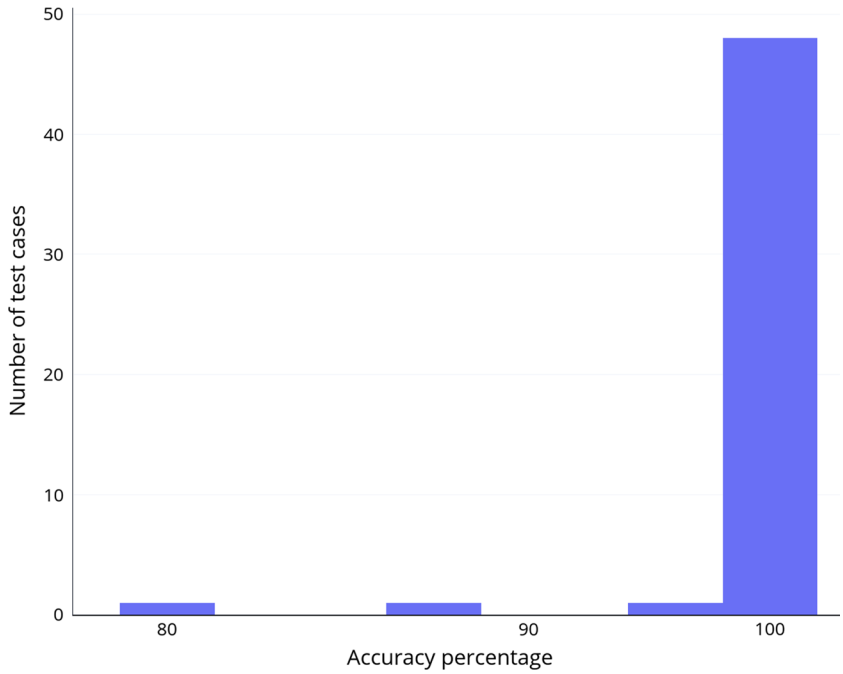

Fig. 11 Histogram for accuracy percentage for different ways of data partitioning, on a synthetic data set of size 1000 × 4

shows the results of these experiments. We conducted a set of 50 tests for each data set, where the data points are reshuffled each time in a different manner before distribution, so that each test assumes different data distributions on workers, where the number of workers is constant. This is achieved by randomly reshuffling the data, but using a different seed each time, which ensures different shuffles. For a small data set with 120 × 2 data points, the accuracy remains 100% regardless of the shuffling. For the 1000 × 4 data set, the accuracy is a high value, close to 100, in the majority of test cases. The lowest accuracy value is 80% here, and it occurs only one time, considering the presented 50 iterations for different shuffles, which represents only 2% of the considered test cases. The obtained results for this data set are also shown in Fig. 11. Regarding the Iris data set, the accuracy is in the range between 86% and 93.33% in all cases, which is also presented in Fig. 12. When the data points are reshuffled, the value of the regularization parameter γ needs to be adjusted in a subset of cases, which is an expected outcome. It can be concluded that the way the input data are partitioned influences the outcomes of the algorithm, to a small extent. However, the level of the accuracy is preserved inside a reasonable range. This trend is a direct consequence of the distributed nature of the algorithm, where the data set is not being consumed as a whole. Hence, the gains of a fast execution time of a parallel algorithm may require a cost of slight accuracy loss that depends on the data distribution.

3.5 Scalability evaluation

In order to evaluate the scaling properties of the parallel ADMM-based convex clustering-like algorithm, different data sets will be used to run on a computer cluster with different numbers of working nodes. All the data sets in these experiments are generated

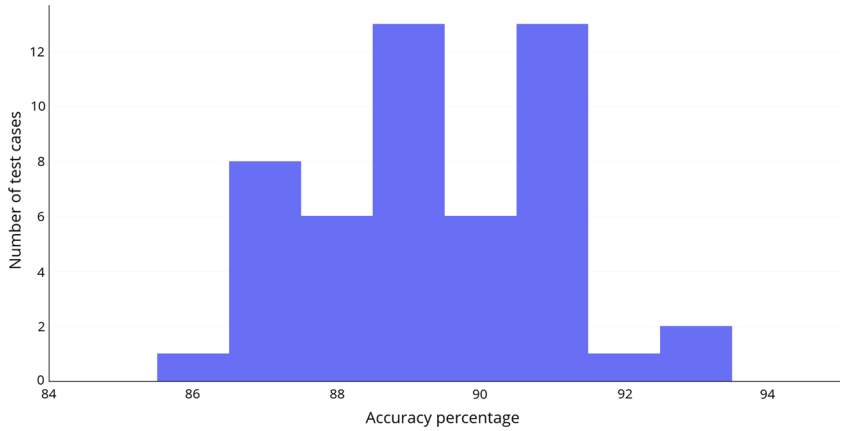

Fig. 12 Histogram for accuracy percentage for different ways of data partitioning, on the Iris data set

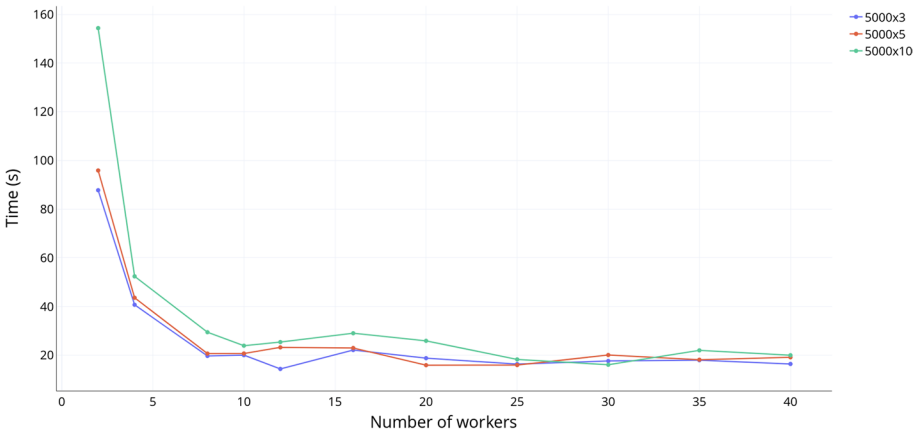

Fig. 13 Scaling properties for the data sets with 5000 samples and 3, 5 and 10 features

as Gaussian mixtures. One aspect is to evaluate run time with respect to the number of workers. Here, we assume a fixed-sized data set is partitioned into an increasing number of workers. Another aspect is to see how the changes in number of features influences the execution. Finally, these tests can provide an insight into the most appropriate number of working nodes for each data set, i.e., the number of nodes that produces the lowest execution time on the data set.[2]

Figures 13 and 14 represent the scaling properties of the algorithm, by showing the execution timings for different data sets and different number of workers.

Figure 13 displays the tests for 3 data sets with 5000 samples and 3, 5 and 10 features. It can be seen that the algorithm scales well and it is straightforward to identify the optimal number of nodes for each data set: 12, 20 and 30, respectively. However, there is a whole range of number of workers where the execution time remains similarly low.

[2] There is always a tradeoff between communication and computation in parallel systems. Splitting the computation to smaller chunks, i.e., adding more nodes, reduces the time required for computation. However, the process of communication/synchronization is becoming more time consuming when increasing the number of nodes. Therefore, an optimal point (a particular number of workers) can be found, where these two aspects are best balanced.

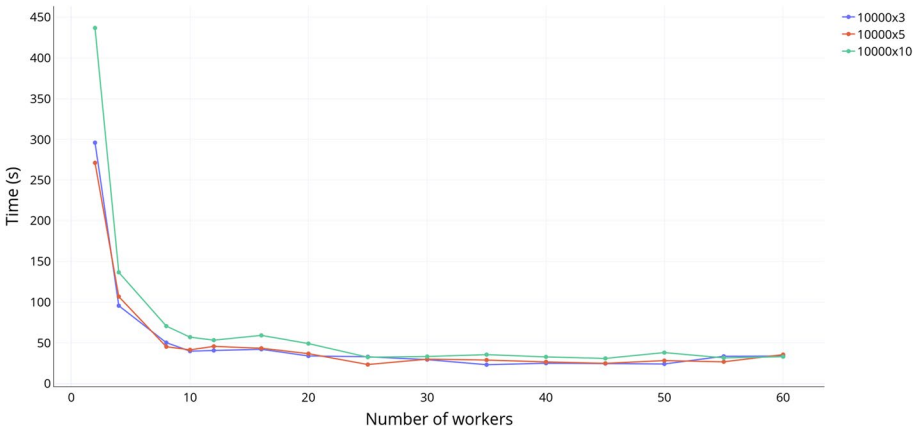

Fig. 14 Scaling properties for the data sets with 10,000 samples and 3, 5 and 10 features

Increasing the number of working nodes reduces the execution time until reaching the optimal range of workers for the data set. The execution time remains approximately the same, within the frames of the conducted experiments. Further increasing the number of workers could lead to execution time increase at some point, as the cost (in terms of communication/synchronization) of having more nodes would then be higher than the gains of parallelization. At some points, the execution time of a larger data set can be slightly lower than the execution time for a smaller data set. The reasons for this could be various, including latency caused by a particular node, but also the nature of the particular data in the data set. It should be kept in mind that measuring the execution time of a synchronous parallel program always subsumes waiting for the slowest process to terminate. Figure 14 shows the scaling properties for 3 data sets with 10, 000 samples and again 3, 5 and 10 features. The algorithm scales well here, as expected. The optimal number of points is 35, 25 and 25, respectively, but the execution time also stays close to this optimal value for a range of different number of workers. It seems strange that we do not have the distribution for optimal points as expected: lower value for less features, higher value for more features. However, this observation is not completely true. The reason is the following: we start with small number of workers and very high execution times. As we increase the number of workers, the execution time rapidly drops, and once it is reduced to certain level, it remains close to that value for further increased number of workers. As a result, the mention optimal number of workers corresponds to the lowest execution time, but that timing is only slightly different for a whole range of tests with different number of nodes.

The displayed experiments showed that the developed algorithm exposes good scaling properties and that the gains of parallelization are evident. The execution time decreases nearly linearly with the number of workers in the experimental range of workers considered.

As the solution of problem (10) depends on the number of workers K, we want to assess how this affects accuracy of the overall proposed clustering method. In order to asses accuracy of the proposed method, when changing the number of workers, we conducted a set of experiments. Figure 15 shows the results of these experiments, for the synthetic 1000×4 data set (generated as a Gaussian mixture model), for fixed values of

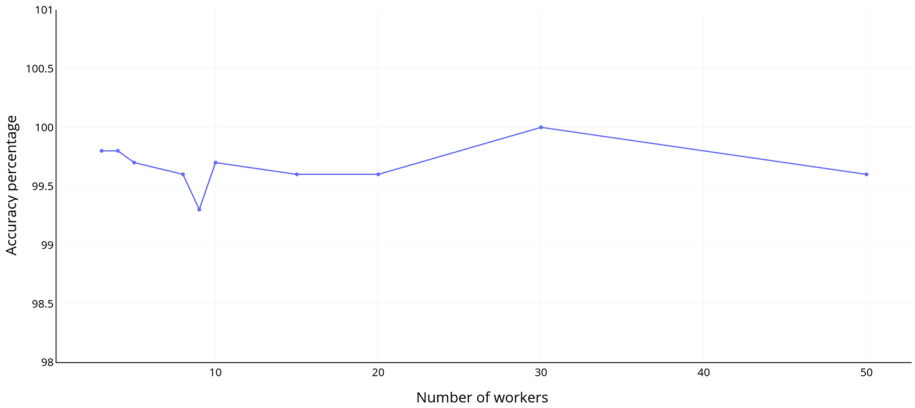

Fig. 15 Accuracy percentage for different number of workers, on the 1000 × 4 data set

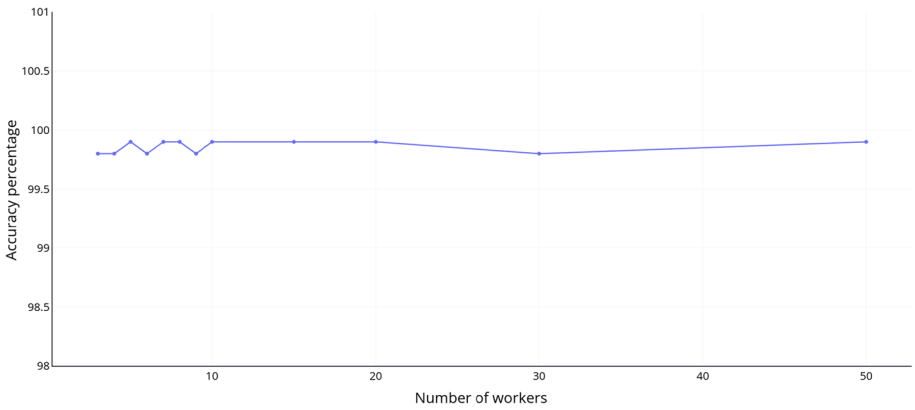

Fig. 16 Accuracy percentage for different number of workers, on the 1000 × 4 data set, with reshuffling

input parameters. It can be seen that the accuracy percentage is being preserved when enlarging the number of workers. It changes slightly, as we change the number of workers, but always remains above 99%. As we already showed that the way we distribute the data may affect the outcome of the algorithm to certain extent, we repeated the tests for different number of workers, with an addition of reshuffling the data, when adding a new worker.

The reshuffling performed here is accomplished in the following way: consider having a sequence of M different numbers of workers $K = 2, \ldots, M$ for the experiments. For $K = 2$, we split the data as already explained, i.e., both workers get a chunk $a_{ij}, i = 1, \ldots, K, j = \frac{N}{K}$. For $K > 2$, we start with the data split for $K - 1$, so that we leave $\frac{N}{K}$ data points (from total $\frac{N}{K-1}$) on the workers from previous run (for $K - 1$), and put the rest to the new worker. This means that the "already existing" workers will have chunks $a_{ij}, i = 1, \ldots K - 1, j = 1, \ldots, \frac{N}{K}$, and the new worker will have the points $a_{ij}, i = 1, \ldots, K - 1, j = \frac{N}{K} + 1, \ldots, \frac{N}{K-1}$. Figure 16 shows the results for this evaluation, for the same data set, with 1000 × 4 data points. The input parameter γ needs to be adjusted, when the number of workers becomes more than 10 in this case, which is a consequence of the data shuffling, that results with changed inter-point distances on a

Table 5 The impact of choosing different values for γ, on a synthetic 30×2 data set

γ	Centroids	Silhouette score
0.3	3	0.89
3.5	3	0.89
5.0	2	0.59
10	2	0.59
25	2	0.54
50	1	0.48

worker. As Fig. 16 shows, the accuracy percentages do not change with enlarged number of workers, even when we reshuffle the data when adding a new worker.

3.6 Choosing the value for the parameter γ

It is well-known that, with SON clustering, parameter γ critically influences performance. The number of clusters for a very small γ equals the number of data points. As γ increases, the number of clusters typically reduces. For gamma above a large threshold, the number of clusters equals one. For all-pair-penalty in [30, 31], SON clustering guarantees to find exact clustering structure, if it exists, for a range of γ. In practice, one can start with a small γ and re-solve the SON problem multiple times, each time increasing γ by a multiplicative functor. This is also known as clusterpath [12]. With our approach, for a fixed ϵ, we observe a similar behavior: the number of clusters reduces when we increase γ, while for a range of γ, we obtain exact recovery (Figs. 4, 6, 7).

We illustrate clusterpath on an example with 30×2 data points, that is suitable for easily plotting the results, as effects of changes in the value of γ. This data set was generated using samples generator from the scikit-learn package [38] that generates isotropic Gaussian blobs for clustering. Table 5 lists the experiments performed for different values of γ. For the value $\gamma = 0.3$, the algorithm results with 3 centers and high silhouette score value. As we increase the regularization parameter, the silhouette score value is getting lower, as well as the number of clusters. For $\gamma = 3.5$, we still have 3 centers, but they are obviously closer to each other than for $\gamma = 0.3$. The silhouette score value is preserved, as the clustering of the data points in this case remains the same as for $\gamma = 0.3$. Increasing the value further as $\gamma = 5.0$, two centers already overlap, resulting with 2 clusters and lower silhouette score value accordingly. Choosing even larger values for γ forces the two centers to become even more close to each other. At the end, for $\gamma = 50.0$, only 1 center remains, as the candidate points for centers overlap.

An approach to choose γ is to evaluate an initial value γ^*, and search in a neighborhood of that value. One way to compute γ^* is as follows:

$$\gamma^* = \max_{i=1\ldots K} \frac{\max_{j \neq l}\{||a_{ij} - a_{il}||\}}{\frac{N}{K}}, \tag{22}$$

Then, we can consider the values $\gamma = \{\frac{\gamma^\star}{100}, \frac{\gamma^\star}{10}, \gamma^\star\, 10 \times \gamma^\star\, 100 \times \gamma^\star\}$.

Table 6 The comparison of execution time (in seconds) for AMA and ADMM-based convex clustering-like method, with the same sub-problems

Data set	Workers	AMA method	ADMM-based convex clustering- like method
1000 × 3	8	0.94	9.73
5000 × 3	25	11.35	14.15
10,000 × 3	25	61.2	22.01
5000 × 5	25	14.5	14.37
10,000 × 10	25	52.63	30.29
200,000 × 3	25	*NA*	564.1

3.7 Comparison with other clustering methods

3.7.1 Comparison with splitting method for convex clustering

In [10], two convex clustering methods are introduced, one based also on ADMM and the other based on alternating minimization algorithm (AMA). The authors provide a rich set of results, based on different tests on synthetic and real data. They clearly identify AMA significantly more efficient. However, the tests described are performed on at most 500 data points, as the subgradient algorithm, used as a benchmark takes a large portion of time to converge on larger data. The R code for these algorithms is available in earlier versions of the CRAN repository.

In order to be able to compare the performance of the AMA method with ADMM-based convex clustering, we measure the execution time of *cvxclust_path_ama*, for only one value of γ, as it corresponds to our setup, where we run our algorithm once for some defined parameter value.

We run a set of tests for the AMA method. The machine used for the tests has 24 GB RAM and an Intel i5-4590 CPU with 4×3.30GHz. We also run ADMM-based convex clustering on the same data sets on our cluster, for different number of workers.

Let us first consider a comparison, where the algorithms solve the same underlying problems. The AMA method can solve problem (10). In order to achieve this, we need to define the weights for AMA, as they are defined in our approach, by the underlying graph. More precisely, we generate the weights for AMA, based on the number of workers, that we use with our algorithm. This weight matrix consists of zeros and ones. Then, we can make a comparison on how efficiently different solvers solve formulation (10). Table 6 shows the comparison of execution times for the AMA method and our approach, when solving the same problems. For smaller data sets, the AMA method performs better, as it represents an efficient serial solver. However, when working on larger data sets, the execution time of our parallel approach can be multiple times lower. Also, as a serial solver, AMA has a limitation on the size of the input data set that can be handled on a single machine. For that reason, the execution of the AMA method on the $200{,}000 \times 3$ data set is not possible. On the other hand, a parallel implementation is only limited by a number of computing units, available on a cluster environment.

In addition to the described comparison of the two solvers of (10) for the same setup, we also want to make a higher level comparison, where we use both methods as proposed, i.e., as complete clustering methods that deliver clustering assignments as

Table 7 The comparison of execution time (in seconds) for AMA and ADMM-based convex clustering methods

Data set	AMA method	ADMM-based conv. clust. like method 4 workers	ADMM-based conv. clust like method 8 workers	ADMM-based conv. clust like method 10 workers	ADMM-based conv. clust. like method 20 workers	ADMM-based conv. clust. like method 25 workers
1000 × 3	2.8	12.65	**9.73**	13.79	11.45	12.24
5000 × 3	17.55	38.38	19.95	16.45	19.35	**14.15**
10,000 × 3	45.46	96.0	51.17	40.84	37.3	**22.01**
5000 × 5	22.74	39.29	19.65	16.87	16.15	**14.37**
10,000 × 10	100.2	136.76	72.63	59.33	50.23	**30.29**
200,000 × 3	N.A.	–	–	–	–	**564.1**

outputs. The aim of this comparison is to compare the final clustering outcomes, and execution times needed to achieve that outcomes. We now consider a comparison of our method with AMA, but without setting the weights in AMA as in our setup. Instead, we use AMA in its standard form, with the recommended weight computation. The results of comparison regarding the Iris data set with known ground truth were already described in Sect. 3.2. The results for other data sets are displayed in Table 7. Note that the lowest execution time for ADMM-based convex clustering-like method is marked in bold, for each data set in Table 7. It can be seen that for data sets of smaller volume, the AMA method can again perform even better than our approach. This is observable for the data set with 1000 × 3 points. Also, the execution times in Table 7 are higher than in Table 6 for the same tests with AMA. This is expected, as the results in Table 6 are for tests that work with sparse weight graph structures. The AMA method, available in R language is actually a wrapper around C code, which is a low level programming language, so the performance is expectably good. Running a smaller example on a cluster naturally does not pay off as in the cases with higher volume of data. For a little larger data set with 5000 × 3 points, the performance of the AMA method is still very close to the performance that can be obtained on the cluster. However, as we increase the volume of the data, it becomes obvious that the ADMM-based convex clustering-like method can perform much better. For the 10,000 × 3 data set, it performs 2 times faster, while for the 10,000 × 10 data set, it performs 3 time faster, when using an appropriate number of workers. Enlarging the data set size even further leads us to cases where the AMA method cannot be run, as it cannot allocate a data structure of the defined volume, due to its serial nature. This is the case for the 200,000 × 3 data set. The AMA method cannot obtain the results, but the ADMM-based convex clustering-like methods solve the problem for 9.4 min with 25 workers. This illustrates the advantage of a parallel approach that can solve large-scale problems.

The AMA method can solve the convex clustering problem with uniform weights. Therefore, we could compare the performance of the AMA method with uniform weights, with our approach. Table 8 illustrates this on the 1000 × 3 data set. The execution of the AMA method is twice slower, than for our approach. A comparison for larger data sets is not feasible, as the AMA method is limited by the memory of a single machine, when allocating large, dense structures. The AMA method's implementation

Table 8 The execution time comparison for ADMM-based convex clustering-like algorithm and for AMA method with uniform weights

Data set	ADMM-based convex clustering-like algorithm	AMA method
1000 × 4	9.73 s	18.19 s

Table 9 Comparison of ADMM-based convex clustering-like method with DBSCAN

Data set	No clust. ADMM	ADMM s.sc.	No clust. k-means	k-means s.sc.	DB- SCAN ϵ	No clust. DB- SCAN	DB- SCAN s.sc.
30 × 2	3	0.89	3	0.89	0.5	3	0.89
40 × 2	6	0.39	5	0.57	0.8	3	0.65
1000 × 3	8	0.76	8	0.76	2.5	8	0.75
5000 × 3	4	0.77	4	0.78	2.5	4	0.76
5000 × 5	5	0.62	4	0.75	5.0	4	0.75
10,000 × 3	10	0.69	10	0.75	2.5	10	0.75

requires the weights specified in an array of size $N \times (N-1)/2$, where N is the number of input data points. It can be concluded that our approach performs better and is more robust than AMA, even when AMA is used with uniform weights.

3.7.2 Comparison with DBSCAN

The ADMM-based convex clustering-like algorithm should also be compared to an other algorithm where the number of clusters is also unknown in advance. We decided to use Density-based spatial clustering of applications with noise (DBSCAN) [43] for this purpose. We use the implementation of DBSCAN from the scikit-learn library. By default, the algorithm uses Euclidean distance as a metric for obtaining the distance values. It accepts a parameter ϵ, representing the maximum distance between two samples for one to be considered as in the neighborhood of the other. This value can be set according to the data set. We set the number of samples in a neighborhood for a point to be considered as a core point to value 2 for all the tests, and run all the tests mentioned before with DBSCAN in order to catch the results for those ϵ values that produce the highest silhouette score.

Table 9 lists the results of experiments, containing the number of clusters and silhouette score for ADMM-based convex clustering-like method, scikit-learn k-means and DBSCAN, respectively. The values for DBSCAN silhouette score assume that there are points labeled as noisy by the algorithm that are not assigned to any of the clusters. For the smallest example (30 × 2 data set), DBSCAN performs in the same manner as our proposed method and k-means, as the number of clusters and the silhouette scores are the same. Let us consider an example of a noisy data set, where the clusters are not clearly separated. For an example of a 40 × 2 data set, where the data are noisy, DBSCAN performs slightly better than our method, resulting with the highest silhouette score value and (expected) 3 clusters, but also leaving some points unlabeled. For the bigger, generated data sets, DBSCAN performs very similar to ADMM-based convex clustering-like method and k-means. Based on these tests, we can conclude that when a clear cluster structure exists, both methods perform satisfactorily.

Table 10 The comparison of SSNAL and ADMM-based convex clustering-like method

SSNAL	ADMM-based convex clustering-like method with 25 workers	ADMM-based convex clustering-like method with 50 workers	ADMM-based convex clustering-like method with 100 workers
3215.79 s + 5390.91 s (for weights)	564.14 s	744.01 s	315.76 s

3.7.3 Comparison with SSNAL method

In [32], a semismooth Newton-based augmented Lagrangian method for solving large-scale convex clustering problems was introduced, called SSNAL. It represents an efficient and robust approach for large-scale problems. The algorithm is developed in MATLAB. A comparison with the AMA method shows great advantage of SSNAL over AMA, in execution time. In order to compare ADMM-based convex clustering-like method with SSNAL, we need a reasonably larger data set, as the performance of a parallel algorithm does not come to expression, when the expenses of parallelization and synchronization are higher than the gains gathered in computation. In order to compare our approach to SSNAL method, let us consider a data set of volume 200,000 $\times$ 3 generated as a Gaussian mixture model. The results are illustrated in Table 10. It can be seen that our proposed approach performs better than SSNAL, even if we consider only the execution time of 321.79 s without computing the weights for SSNAL. It should also be noted that the presented time for our method also includes the time spent on merging the cluster centers when the algorithm terminates.

As the comparison above showed, the computation of weights for the SSNAL method requires a tremendous amount of time. We expect that SSNAL cannot scale as effectively as the proposed approach on larger data sets due to the serial implementation and calculation of weights that are pair-wise across all pairs of data in the data set. To further demonstrate this, we evaluated execution times for weights calculation for data sets of different sizes. Note that the weights calculation time represents a lower bound on the execution time of the overall SSNAL method. Therefore, if the execution time of the proposed method is smaller than that of the weight calculation, it follows that the execution time of the proposed method is smaller than that of SSNAL overall. We also want to investigate the cost of the weight assignment, when implemented as in [32] and measure the time required for this kind of preprocessing. Assigning weights according to the nature of the data is very useful, but the process has a certain cost. We want to identify how much time is needed for this kind of preprocessing, in order to compare it to our execution time. The calculation of the weights, as stated in [32], can be done as follows:

$$w_{ij} = \begin{cases} \exp(-0.5||a_i - a_j||^2 & if\,(i,j) \in E, \\ 0 & \text{otherwise} \end{cases} \tag{23}$$

Here, $E = U_{i=1}^{N}\{(i,j)|a_j$ is among a_i's k nearest neighbors, $i < j \leq N\}$. This kind of preprocessing is not applicable in this form to ADMM-based convex clustering-like method, due to its distributed nature. Computing the pairwise distances among points that are assigned to different workers could be very expensive. However, it is indisputable that assigning weights could seriously affect the performance of the algorithm afterwards. Therefore, we analyze the execution time required for obtaining the weights, as well as

Table 11 The execution time required for obtaining the weights

Data set size	*k* for KNN	Time
200,000 × 3	10	24.4 s
200,000 × 3	100	61.38 s
200,000 × 3	1000	959.32 s
2,000,000 × 3	10	18.46 min
2,000,000 × 3	100	59.05 min

Table 12 The execution time comparison for ADMM-based convex clustering-like algorithm and for SSNAL method with uniform weights

Data set	ADMM-based convex clustering-like algorithm	SSNAL method with uniform weights total time	SSNAL method with uniform weights time for weights setup
1000 × 4	9.73 s	230.79 s	191.12 s

the execution time of ADMM-based convex clustering-like method itself. We wrote a Python script that works sequentially with sparse data structures, and tested it for 2 data sets and a few different values of k, used for k-nearest neighbors.

The execution time for different values of k, for the mentioned 2 data sets, is shown in Table 11. For the smaller data set with 200,000 × 3 data points and $k = 10$ and $k = 100$, the required time for weights calculation is low, compared to the overall execution time of 9.4 min on the cluster. However, when increasing k to $k = 1000$, the time grows to 15 min, that is longer than the time required to solve the problem by ADMM-based convex clustering-like method on the cluster. Considering the larger data set, with 2,000,000 × 3 data points, the execution time for weight calculation is extensive. For $k = 100$, it takes almost an hour. Setting the parameter k for KNN is always an open issue. However, for a data set of large volume, as the data sets displayed here, it is likely that a larger value of k will be needed. This could result with a very time consuming preprocessing step that can actually be higher than the execution time for ADMM-based convex clustering-like method on a cluster that does not use any preprocessing. Our approach uses pairwise distances only within workers, when the value of ϵ is being computed, for the merging process.

Similarly as the AMA method, the SSNAL method can solve the convex clustering problem, by using uniform weights. Let us consider an example. We compare the execution time for our approach and the SSNAL method in Table 12. It displays the results for the 1000 × 4 data set. The execution time required for setting up the weights with SSNAL method is still high, as it represents an allocation of a large matrix that is dense. This could be completely bypassed by replacing the multiplications with elements from the weights matrix with ones in the code directly. However, we are interested in the amount of time, required for the rest of the computation. It can be easily seen from Table 12, that the time required for the algorithm only is 39.67 s, which is a couple of times more than the execution time for our method. This comparison cannot be made for larger data sets, as the large dense weight matrices deplete the resources of the machine, for SSNAL method. It could be possibly achieved by completely removing

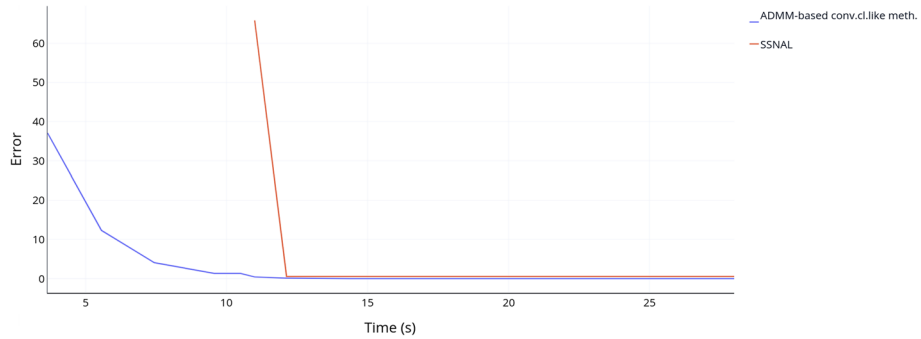

Fig. 17 A comparison of solvers for ADMM-based convex clustering-like method and SSNAL method

the matrix from the code, as already stated, but this is left for future comparison, as the main trend is already presented with the 1000×4 data set.

Various methods utilize different stopping criteria that are adjusted to particular algorithms. Besides comparing different clustering approaches with their own stopping conditions, we also want to make a comparison, regardless of those stopping criteria. We illustrate this for SSNAL method. The aim is to investigate how the different solvers for ADMM-based convex clustering-like method and SSNAL algorithm relate. Therefore, we let the algorithms run without stopping conditions for the same value of the regularization parameter γ, while recording the execution time and the current solution, as the iterations progress. This means that we use a predefined large, fixed number of iterations for both methods, ensuring high solution accuracy. In order to make a fair comparison, we initialize the solutions to all zeros and generate a weight matrix of zeros and ones for SSNAL method that corresponds to our SON penalty structure as in formulation (10) for a particular number of workers. For each computed solution update, we determine the error, by computing the Euclidean distance to a benchmark solution, obtained by solving (10), with CVXPY. We illustrate these results for a synthetic data set containing 1000×4 data points (generated as a Gaussian mixture model), used with 5 workers, in Fig. 17. Note that the methods do not have the same staring point, as we show the error values after the first results are obtained. Figure 17 shows that both methods reduce the error over time. For our method, there is a gradual decrease in error rate during the iterations, while SSNAL tends to consume a larger portion of time at the beginning, while setting up the weights and performing the first phase of the algorithm, meant for warm start. Evidently, our method approaches the neighborhood of the solution sooner, even for the described, relatively small volume of the problem.

3.8 Further implementation considerations

The current implementation of the proposed method utilizes CVXPY to solve sub-problems, and it can be set to use the so-called warm start option. Generally, improvements in execution time can be expected if we use a "warm start" in updates (14), (15) and (16) and initialize the new variables with their values from the previous iteration. In addition, as commonly used with ADMM, the sub-problems (12) may not be solved to full accuracy, i.e., they can be solved inexactly. (Theoretically, the accuracy of solving (12) should be increasing with the iterations counter for exact convergence). We examine these two

Table 13 The impact of solver enhancement on performance

Data set	No enhancement	Warm start	Accuracy adjustment	Warm start and accuracy adjustment
10,000 × 3	38.65 s	37.5 s	37.7 s	35.88 s

strategies on the CVXPY implementation. This way the solver starts with the solution from the previous iteration, which can reduce the execution time. Another approach is to set a solver property such that the expected accuracy becomes lower during the first few iterations. The ECOS solver, used by CVXPY in this case, has the property *abstol*, which corresponds to absolute accuracy tolerance, and it has a default value of $1e{-}8$. By enlarging this value, we permit greater difference, i.e., lower accuracy. We tested this approach by using 0.0001 during the first 3 iterations of the algorithm.

Let us illustrate the impacts of these enhancements to performance. Table 13 shows the execution time for different variants of the algorithm: without enhancement, with warm start, with tolerance set up and with both warm start and tolerance set up. Apparently, these enhancements can reduce the execution time to certain extent. For the data set with 10,000 × 3 points, the time reduction is 2.7 s, i.e., 7% roughly. This is not a drastic difference, but it certainly represents an improvement in performance.

In addition, subproblems (12) can be solved by adopting efficient moderate-size problem SON clustering solvers like [26], instead of using the general-purpose solver like CVXPY. This approach can expose equally good or even better performance than CVXPY. For example, some initial tests show that the data set with 10,000 × 3 points (generated as a Gaussian mixture model) can be solved for 6.7 s with 25 workers, where with the CVXPY solver, the same problem was solved for 22 s, with the same number of workers. The accuracy of the solution is the same as for CVXPY. Therefore, this approach represents a promising direction for further enhancement.

4 Conclusions

In this paper, we introduce a parallel ADMM-based convex clustering-like algorithm and provide a parallel implementation for it, supported by a wide set of test cases. The comprehensive empirical evaluations prove that the algorithm satisfies a similar level of accuracy as the other widely used clustering approaches. It was also shown that the algorithm can work with large data sets efficiently, exhibiting good scaling properties on a cluster environment.

The tests were performed on an HPC computer cluster, AXIOM. The configuration of the AXIOM computing facility consists of 16 nodes, where each node has a processor with 6 CPU cores (eighth-generation Core i7 cores). The nodes are connected by an Ethernet network with speed of 10 Gbps. The described behavior of the algorithm during the tests should be preserved when tested on other cluster environment. The execution time may be shorter on a cluster with newer generation of processors, but the overall performance characteristics as the scaling properties and the advantages of parallel execution are expected to be the same. A higher network speed is expected to produce good performance with more nodes than in our experiments, so the range of the number of nodes with lowest execution time would be different, but still detectable.

There are several possibilities for expanding the idea of the ADMM-based convex clustering-like algorithm. For instance, in order to enhance further the algorithm speed, efficient sub-problem solvers could be further designed. In addition, data-dependent sparse graph construction for (14)–(15) and weighted sparse SON penalty can be considered with an additional preprocessing cost.

Abbreviations

COMPSs	COMPS superscalar
HPC	High-Performance Computing
I/O	Input/Output
ADMM	Alternating Direction Method of Multipliers
SON clustering	Sum Of Norms clustering

Acknowledgements
The work of this paper has been carried out within EU Project CYRENE, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 952690. This publication reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein. The authors gratefully acknowledge the AXIOM HPC facility at Faculty of Sciences, University of Novi Sad, where all the numerical simulations were run.

Author contributions
LF developed the implementation of the algorithm and performed the empirical evaluations. DJ contributed with the theoretical advances and design of algorithm. DBK and SS contributed to improving the quality of experimentation and design. All authors participated in the main research flow development and in writing and revising the manuscript. All authors read and approved the final manuscript.

Funding
Not applicable.

Availability of data and materials
The code for parallel ADMM-based convex clustering can be found in the following GitHub repository: https://github.com/lidijaf/Parallel-ADMM-based-convex-clustering. The datasets used and analyzed during the current study are available from the corresponding author on reasonable request.

Declarations

Competing interests
The authors declare that they have no competing interests.

Received: 21 June 2022 Accepted: 24 October 2022
Published online: 08 November 2022

References

1. T. Warren Liao, Clustering of time series data—a survey. Pattern Recogn. **38**(11), 1857–1874 (2005). https://doi.org/10.1016/j.patcog.2005.01.025
2. X. Dai, T. Kuosmanen, Best-practice benchmarking using clustering methods: application to energy regulation. Omega **42**(1), 179–188 (2014). https://doi.org/10.1016/j.omega.2013.05.007
3. T. Chaira, A novel intuitionistic fuzzy C means clustering algorithm and its application to medical images. Appl. Soft Comput. **11**(2), 1711–1717 (2011). https://doi.org/10.1016/j.asoc.2010.05.005
4. F. Lindsten, H. Ohlsson, L. Ljung, Clustering using sum-of-norms regularization: with application to particle filter output computation, in *2011 IEEE Statistical Signal Processing Workshop (SSP)* (2011), pp. 201–204. https://doi.org/10.1109/SSP.2011.5967659
5. S. Lloyd, Least squares quantization in PCM. IEEE Trans. Inf. Theory **28**(2), 129–137 (1982). https://doi.org/10.1109/TIT.1982.1056489
6. D. Arthur, S. Vassilvitskii, K-means++: the advantages of careful seeding, in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms. SODA '07* (Society for Industrial and Applied Mathematics, USA, 2007), pp. 1027–1035
7. S. Arora, P. Raghavan, S. Rao, Approximation schemes for Euclidean k-medians and related problems, in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing. STOC '98* (Association for Computing Machinery, New York, NY, USA, 1998), pp. 106–113. https://doi.org/10.1145/276698.276718
8. A. Banerjee, S. Merugu, I.S. Dhillon, J. Ghosh, Clustering with Bregman divergences. J. Mach. Learn. Res. **6**(58), 1705–1749 (2005). https://doi.org/10.1137/1.9781611972740.22
9. F. Lindsten, H. Ohlsson, L. Ljung, Just relax and come clustering!: a convexification of k-means clustering (2011). http://liu.diva-portal.org/smash/get/diva2:650707/FULLTEXT01.pdf

10. E.C. Chi, K. Lange, Splitting methods for convex clustering. J. Comput. Graph. Stat. **24**(4), 994–1013 (2015). https://doi.org/10.1080/10618600.2014.948181
11. K. Pelckmans, J.D. Brabanter, B.D. Moor, J.A.K. Suykens, Convex clustering shrinkage, in *Workshop on Statistics and Optimization of Clustering Workshop (PASCAL)* (2005)
12. T.D. Hocking, A. Joulin, F. Bach, J.-P. Vert, Clusterpath: an algorithm for clustering using convex fusion penalties, in *Proceedings of the 28th International Conference on International Conference on Machine Learning. ICML'11* (Omnipress, Madison, WI, USA, 2011), pp. 745–752
13. S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends Mach. Learn. **3**(1), 1–122 (2011). https://doi.org/10.1561/2200000016
14. Parallel ADMM-based convex clustering. https://github.com/lidijaf/Parallel-ADMM-based-convex-clustering. Accessed on 10 June 2022
15. E. Tejedor, Y. Becerra, G. Alomar, A. Queralt, R.M. Badia, J. Torres, T. Cortesand, J. Labarta, PyCOMPSs: parallel computational workflows in python. Int. J. High Perform. Comput. Appl. **31**(1), 66–82 (2017). https://doi.org/10.1177/1094342015594678
16. H. Steinhaus, Sur la division des corps matériels en parties. Bull. Acad. Pol. Sci., Cl. III **4**, 801–804 (1957)
17. J.M. Peña, J.A. Lozano, P. Larrañaga, An empirical comparison of four initialization methods for the k-means algorithm. Pattern Recognit. Lett. **20**(10), 1027–1040 (1999). https://doi.org/10.1016/S0167-8655(99)00069-0
18. S.S. Khan, A. Ahmad, Cluster center initialization algorithm for k-means clustering. Pattern Recognit. Lett. **25**(11), 1293–1302 (2004). https://doi.org/10.1016/j.patrec.2004.04.007
19. D. Lashkari, P. Golland, *Convex Clustering with Exemplar-Based Models* (MIT Press, Cambridge, MA, 2007), pp.825–832
20. S. Nowozin, G. Bakir, A decoupled approach to exemplar-based unsupervised learning, in *Proceedings of the 25th International Conference on Machine Learning. ICML '08* (Association for Computing Machinery, New York, NY, USA, 2008), pp. 704–711. https://doi.org/10.1145/1390156.1390245
21. M. Wang, T. Yao, G.I. Allen, Supervised convex clustering. arXiv preprint arXiv:2005.12198 (2020)
22. G.K. Chen, E.C. Chi, J.M. Ranola, K. Lange, Convex clustering: an attractive alternative to hierarchical clustering. PLoS Comput. Biol. **11**(5), e1004228 (2015). https://doi.org/10.1371/journal.pcbi.1004228
23. B. Wang, Y. Zhang, W. Sun, Y. Fang, Sparse convex clustering. J. Comput. Graph. Stat. **27**(2), 393–403 (2018). https://doi.org/10.1080/10618600.2017.1377081
24. M. Yuan, Y. Lin, Model selection and estimation in regression with grouped variables. J. R. Stat. Soc. B **68**, 49–67 (2006). https://doi.org/10.1111/j.1467-9868.2005.00532.x
25. Q. Qian, On algorithmic regularization and convex clustering. PhD thesis, SOhio State University. OhioLINK Electronic Theses and Dissertations Center (2019). http://rave.ohiolink.edu/etdc/view?acc_num=osu156594573920941
26. Y.L. Huangyue Chen, Lingchen Kong, A novel convex clustering method for high-dimensional data using semiproximal ADMM. Math. Probl. Eng. **2020**, Article ID 9216351, 12 (2020)
27. S. Kar, B. Swenson, Clustering with distributed data (2019). https://doi.org/10.48550/ARXIV.1901.00214. arXiv:1901.00214
28. W. Zhou, H. Yi, G. Mishne, E. Chi, Scalable algorithms for convex clustering, in *2021 IEEE Data Science and Learning Workshop (DSLW)* (2021), pp. 1–6. https://doi.org/10.1109/DSLW51110.2021.9523411
29. X. Zhou, C. Du, X. Cai, An Efficient Smoothing Proximal Gradient Algorithm for Convex Clustering. arXiv (2020). https://doi.org/10.48550/ARXIV.2006.12592. arXiv:2006.12592
30. C. Zhu, H. Xu, C. Leng, S. Yan, Convex optimization procedure for clustering: theoretical revisit, in *Advances in Neural Information Processing Systems*, vol. 27, ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, K.Q. Weinberger (Curran Associates Inc., New York, 2014), pp.1619–1627
31. A. Panahi, D. Dubhashi, F.D. Johansson, C. Bhattacharyya, Clustering by sum of norms: stochastic incremental algorithm, convergence and cluster recovery, in *Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research*, vol. 70, ed. by D. Precup, Y.W. Teh (PMLR, Sydney, 2017), pp.2769–2777
32. D. Sun, K.-C. Toh, Y. Yuan, Convex clustering: model, theoretical guarantee and efficient algorithm. J. Mach. Learn. Res. **22**, 9–1932 (2021)
33. R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, K. Knight, Sparsity and smoothness via the fused lasso. J. R. Stat. Soc.: Ser. B (Stat. Methodol.) **67**(1), 91–108 (2005). https://doi.org/10.1111/j.1467-9868.2005.00490.x
34. Y. Gdalyahu, D. Weinshall, M. Werman, A randomized algorithm for pairwise clustering, in *NIPS* (1998)
35. S. Diamond, S. Boyd, CVXPY: a python-embedded modeling language for convex optimization. J. Mach. Learn. Res. **17**(83), 1–5 (2016)
36. A. Agrawal, R. Verschueren, S. Diamond, S. Boyd, A rewriting system for convex optimization problems. J. Control Decis. **5**(1), 42–60 (2018)
37. E. Tejedor, Y. Becerra, G. Alomar, A. Queralt, R.M. Badia, J.T. Toni Cortes, J. Labarta, PyCOMPSs: parallel computational workflows in Python. IJHPCA **31**(1), 66–82 (2017). https://doi.org/10.1177/1094342015594678
38. scikit-learn: Machine learning in Python. https://scikit-learn.org/stable/. Accessed on 20 March 2022
39. C. Rasmussen, The infinite gaussian mixture model, in *Advances in Neural Information Processing Systems*, vol. 12, ed. by S. Solla, T. Leen, K. Müller (MIT Press, Cambridge, MA, 1999)
40. L. van der Maaten, G. Hinton, Visualizing data using t-SNE. J. Mach. Learn. Res. **9**, 2579–2605 (2008)
41. E. Anderson, The species problem in Iris. Ann. Mo. Bot. Gard. **23**(3), 457–509 (1936)
42. R.A. Fisher, The use of multiple measurements in taxonomic problems. Ann. Hum. Genet. **7**, 179–188 (1936)
43. M. Ester, H.-P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining. KDD'96* (AAAI Press, Palo Alto, CA, 1996), pp. 226–231

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.