
Dynamic Visualization for L1 Fusion Convex Clustering in Near-Linear Time

Bingyuan Zhang¹

Jie Chen¹

Yoshikazu Terada^{1,2}

¹Graduate School of Engineering Science, Osaka University, Japan.

²RIKEN Center for Advanced Intelligence Project (AIP)

Abstract

Convex clustering has drawn recent attention because of its competitive performance and nice property to guarantee global optimality. However, convex clustering is infeasible due to its high computational cost for large-scale data sets. We propose a novel method to solve the L_1 fusion convex clustering problem by dynamic programming. We develop the Convex clustering Path Algorithm In Near-linear Time (C-PAINT) to construct the solution path efficiently. The proposed C-PAINT yields the exact solution while other general solvers for convex problems applied in the convex clustering depend on tuning parameters such as step size and threshold, and it usually takes many iterations to converge. Including a sorting process that almost takes no time in practice, the main part of the algorithm takes only linear time. Thus, C-PAINT has superior scalability comparing to other state-of-art algorithms. Moreover, C-PAINT enables the path visualization of clustering solutions for large data. In particular, experiments show our proposed method can solve the convex clustering with 10^7 data points in \mathbb{R}^2 in two minutes. We demonstrate the proposed method using both synthetic data and real data. Our algorithms are implemented in the `dpcc` R package.

1 INTRODUCTION

Clustering is one of the most popular unsupervised learning tasks exploring data and seeking groups of similar objects. Traditional clustering methods include hierarchical clustering, partitive clustering, and model-based clustering. Recently, convex clustering has been studied [Hocking et al., 2011, Lindsten et al., 2011, Pelckmans et al., 2005], which guarantees global optimality due to the convex formulation

of the problem. Different from the methods like k -means that requires a given cluster number, convex clustering uses a tuning parameter to control the number of output clusters.

Given n points $\mathbf{x}_1, \dots, \mathbf{x}_n$ in \mathbb{R}^p , convex clustering minimizes the following problem:

$$L(\mathbf{A}) = \frac{1}{2} \sum_{i=1}^n \|\mathbf{x}_i - \mathbf{a}_i\|_2^2 + \lambda \sum_{i < j} \omega_{ij} \|\mathbf{a}_i - \mathbf{a}_j\|_q \quad (1)$$

where each $\mathbf{a}_i \in \mathbb{R}^p$ is the i -th row of the matrix \mathbf{A} that provides an alternative vector to represent the point \mathbf{x}_i . $\|\cdot\|_q$ denotes the L_q -norm, typically chosen 1, 2, or ∞ . λ is a positive tuning parameter, and ω_{ij} are the given weights that are generally chosen based on the given input data. After solving the optimization problem, we obtain the optimal solution $\hat{\mathbf{A}} = (\hat{\mathbf{a}}_1, \dots, \hat{\mathbf{a}}_n)^T$. We assign the sample i and j to a same cluster if and only if $\hat{\mathbf{a}}_i = \hat{\mathbf{a}}_j$.

The optimization problem also has a meaningfully visualized interpretation called “clusterpath” (Figure 1) when varying the tuning parameter λ . With $\lambda = 0$, each point occupies a unique cluster, as λ increases, the clusters begin to coalesce. In the end all the points coalesce into a single cluster for a sufficiently big λ . The clusterpath shows how each point becomes merged along the path with different λ s, and the visualization provides rich information about the cluster structure of data.

In general, the computational cost to construct the clusterpath is very high. In order to solve (1), Hocking et al. [2011] introduced three different algorithms for different regularizers corresponding to L_1, L_2 and L_∞ . After that, general solver for convex problem such as the alternating direction method of multipliers (ADMM) and the alternating minimization algorithm (AMA) [Chi and Lange, 2015] are also applied to solve (1) with L_1 and L_2 penalties. In order to obtain the clusterpath efficiently, Weylandt et al. [2020] proposed CARP algorithm which uses an novel computational technique to approximate the path-wise visualizations with sufficient precision. Radchenko and Mukherjee [2017] considered two efficient algorithms for L_1 penalty case that

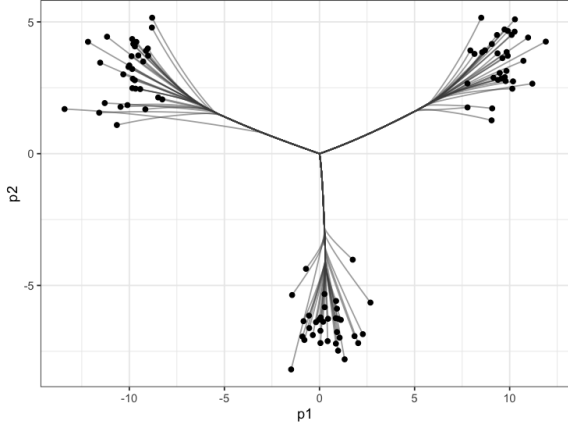


Figure 1: An example shows the clusterpath generated under L_1 norm. The clusterpath shows individual cluster centers start to merge and finish as a single cluster.

successively merges the clusters in a bottom-up fashion or splits the clusters in a top-down fashion to detect all the fusion or split events. Moreover, Radchenko and Mukherjee [2017] studied the sample behavior of convex clustering with L_1 penalty and identical weights and provided theoretical support. However, their methods cannot estimate $\hat{\mathbf{A}}$ and thus cannot provide a clusterpath.

In this paper, we consider the same setting as Radchenko and Mukherjee [2017]: L_1 convex clustering with identical weights. We consider a completely different approach and develop an efficient algorithm to handle the computational bottleneck in the convex clustering problem. Fortunately, for the problem (1) with the L_1 penalty, Hocking et al. [2011] noted that the problem was separable on dimensions. In addition, for each dimension the problem can be considered as a fused lasso problem [Tibshirani et al., 2005]. The problem (1) is decomposed into p separate sub-problems as follows:

$$\min_{\mathbf{a} \in \mathbb{R}^n} \frac{1}{2} \sum_{i=1}^n (x_i - a_i)^2 + \lambda \sum_{i=1}^{n-1} \sum_{j=i+1}^n |a_i - a_j| \quad (2)$$

Hocking et al. [2011] adapted the FLSA algorithm [Hoeffling, 2010] to solve the problem (2). Nevertheless, based on our experience, it still remains very challenging for large scale problems. Thus we employ a way of dynamic programming (DP) method to obtain the exact solution of the problem (2). Johnson [2013] first proposed a DP method for the chain graph fused lasso, or simply, 1d fused lasso problem which penalizes the neighbor terms:

$$\min_{\mathbf{a} \in \mathbb{R}^n} \frac{1}{2} \sum_{i=1}^n (x_i - a_i)^2 + \lambda \sum_{i=1}^{n-1} |a_i - a_{i+1}| \quad (3)$$

On the surface, (2) is not equivalent to (3). In this paper, we first show the sub-problem of convex clustering (2) can be

reformulated into a weighted one-dimensional fused lasso problem. The transformation allows us to apply a modified version of the DP method to solve the problem in linear time. The numerical experiments show the proposed method yields superior performance. We summarize our main contributions:

- We show that each sub-problem of convex clustering can be reformulated into a weighted 1d fused lasso problem based on an important observation result for identical weights case.
- We employ an DP algorithm to solve the reformulated problem. Furthermore, due to the special formulation of the problem, we refine it to be more efficient.
- We propose an algorithm named C-PAINT based on the DP algorithm to construct a full clusterpath. The time complexity of C-PAINT is $\mathcal{O}(pn \log n) + \mathcal{O}(pnK)$, where K is the length of λ sequence, n is the sample size, and p is the feature dimension of each sample. In practice, we show the C-PAINT takes $\mathcal{O}(pnK)$ which is scalable to large dataset.

The remaining article is arranged as follows. Related work is introduced in Section 2. Section 3 provides the preliminaries and introduces some properties of the convex clustering to be used to reformulate problem (2) later. In Section 4, we present the DP method and C-PAINT algorithm to draw a full clusterpath. The experimental results are reported in Section 5, including both synthetic data and real data. At last, Section 6 concludes the article. In addition, we provide the details about the DP algorithm in the supplementary material.

2 RELATED WORK

There are some variants of convex clustering. Chi et al. [2017] considered the convex bi-clustering. Wang et al. [2018] proposed the sparse convex clustering which can perform clustering and feature selection simultaneously. The robust convex clustering [Wang et al., 2016] was proposed to detect the outlier features.

From the theoretical perspective, Tan and Witten [2015] showed that when the identical weights are used, the convex clustering has a close connection to the single-linkage hierarchical clustering. Radchenko and Mukherjee [2017] analysed the asymptotic properties of the solution path and gave conditions for it to yield the true dendrogram under L_1 fusion penalty with identical weights. Zhu et al. [2014] studied the condition for convex clustering to recover the clusters correctly.

From the computational perspective, Lindsten et al. [2011] proposed to use the off-the-shelf solver called CVX to generate the solution path. Hocking et al. [2011] introduced three algorithms for three different penalty norms

(L_1, L_2 , and L_∞). Especially, they used the FLSA algorithm for L_1 penalties. Chi and Lange [2015] proposed the ADMM and the AMA for the convex clustering problem. However, the convergence rate is not fast enough during the iterative process when the sample size n and the dimension of data p are large. Yuan et al. [2018] proposed a semismooth Newton based algorithm to solve the convex clustering problem. Radchenko and Mukherjee [2017] proposed efficient methods that successively merge or split the clusters, but these methods are unable to find the exact solution of the estimated centroids $\hat{\mathbf{A}}$, thus is impossible for the visualization of the clusterpath.

Although convex clustering has many competitive features, its computational burden remains to be challenging.

3 PRELIMINARIES

In this paper, we use bold small letters to represent vectors like \mathbf{x} , and ordinary one for scalar like x .

We first review the fact that the convex clustering problem can be decomposed into sub-problems by dimension. Then we go through some important facts about the clusterpath, which prepares us to reformulate the problem in Section 4.

Each sample point has p features $\mathbf{x}_i = (x_{i1}, \dots, x_{ip})^T$ and its corresponding parameter vectors $\mathbf{a}_i = (a_{i1}, \dots, a_{ip})^T$.

Consider the convex clustering problem with L_1 fusion penalty and identical weights:

$$\begin{aligned} & \frac{1}{2} \sum_{i=1}^n \|\mathbf{x}_i - \mathbf{a}_i\|_2^2 + \lambda \sum_{i < j} \|\mathbf{a}_i - \mathbf{a}_j\|_1 \\ & = \sum_{k=1}^p \left[\frac{1}{2} \sum_{i=1}^n (x_{ik} - a_{ik})^2 + \lambda \sum_{i=1}^{n-1} \sum_{j=i+1}^n |a_{ik} - a_{jk}| \right] \end{aligned}$$

Thus solving the minimization problem of (1) just amounts to solving p separate sub-problems on each dimension. In the following content, we consider the sub-problem (2).

We now introduce the following theorem and lemma that prepare us for the reformulation in the next section. The theorem 3.1 shows that in the L_1 clusterpath, no split happens in the identical weights setting. Thus, we have lemma 3.2 that the order of the original input are preserved in estimated centroids.

Theorem 3.1 (Hocking et al. [2011]). *Taking $\omega_{ij} = 1$ for all i and j is sufficient to ensure that the L_1 clusterpath contains no splits.*

Lemma 3.2 (Chiquet et al. [2017]). *The absence of splits is equivalent to preservation of the order along the path for problem (2).*

4 PROPOSED METHODS

In this section, we first show how to reformulate the problem (2) by substituting with the penalty term in Section 4.1. Next, after the reformulation of the objective function, we can adapt the DP method to solve it. We explain the details about the modified DP algorithm in Section 4.2. Algorithm 1 gives a high-level view of the DP algorithm, while further details are presented in algorithm 2. The C-PAINT algorithm is based on the DP algorithm and presented in algorithm 3 in Section 4.3. Finally, we give some analysis about the time complexity in Section 4.4.

4.1 IDEAS

On the theoretical side, a direct consequence of theorem 3.1 and lemma 3.2 is: for (2), the order of \mathbf{x} are preserved, in other words, the estimated centroids preserve the original order of the input data.

$$x_{(1)} \leq x_{(2)} \leq \dots \leq x_{(n)} \quad \longrightarrow \quad \hat{a}_{(1)} \leq \hat{a}_{(2)} \leq \dots \leq \hat{a}_{(n)}$$

where the $x_{(i)}, i = 1, \dots, n$ are the \mathbf{x} that sorted in a non-decreasing order. Thus the order of the centroids to be estimated can be obtained directly from the input data. Let us take a look at the penalty term in problem (2): $\sum_{i=1}^{n-1} \sum_{j=i+1}^n |a_i - a_j|$. Suppose the order of (a_1, \dots, a_n) is known, then the absolute value can be removed. For example for $a_{(1)} \leq a_{(2)} \leq a_{(3)} \leq a_{(4)}$, the absolute value of $|a_{(1)} - a_{(3)}| = a_{(3)} - a_{(1)} = a_{(3)} - a_{(2)} + a_{(2)} - a_{(1)}$, which can be written as $|a_{(2)} - a_{(3)}| + |a_{(1)} - a_{(2)}|$. By decomposing penalty terms, we can rewrite it into the absolute values of the differences between neighbor items. The Figure 2 shows an image of the transformation from a complete graph into a weighted chain graph.

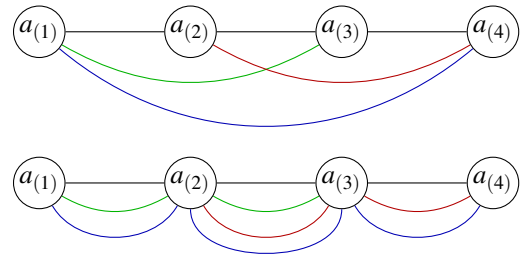


Figure 2: Transformation of the penalty graph. The edges are the absolute values of the differences between nodes. The total sums of the edges length are identical for two graphs, which inspires us to reformulate the penalty term.

Lemma 4.1. *Given the sequence (a_1, \dots, a_n) , sort it in a non-decreasing order $a_{(1)} \leq \dots \leq a_{(n)}$, then*

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^n |a_i - a_j| = \sum_{i=1}^{n-1} i(n-i) |a_{(i)} - a_{(i+1)}|. \quad (4)$$

Lemma 4.1 suggests that it is possible to replace the penalty term in (2) with the right side one in (4). This reformulation turns the fused lasso problem into a weighted 1d fused lasso problem as follows:

$$\min_{\mathbf{a} \in \mathbb{R}^n} \frac{1}{2} \sum_{i=1}^n (x_i - a_i)^2 + \lambda \sum_{i=1}^{n-1} i(n-i) |a_{(i)} - a_{(i+1)}| \quad (5)$$

Now we can employ the DP [Johnson, 2013] to solve this problem (5). In addition, we know $\hat{a}_{(i)} \leq \hat{a}_{(i+1)}$ always hold for i , thus we only need to consider the cases $\hat{a}_{(i)} = \hat{a}_{(i+1)}$ and $\hat{a}_{(i)} < \hat{a}_{(i+1)}$, which refines the DP algorithm to be more efficient. Next, we introduce the modified DP algorithm in further detail.

4.2 DP ALGORITHM

Given the data points $x_1 \leq x_2 \leq \dots \leq x_n$, by lemma 3.2, the order are preserved for the centroids $\hat{a}_1 \leq \hat{a}_2 \leq \dots \leq \hat{a}_n$, we consider the following problem:

$$(\hat{a}_1, \dots, \hat{a}_n) := \underset{\hat{\mathbf{a}}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^n (x_i - a_i)^2 + \sum_{i=1}^{n-1} \lambda_i |a_i - a_{i+1}|$$

Before giving details about the DP algorithm, it is necessary to prepare some notation here. Let

$$h_1(b) := \frac{1}{2} (x_1 - b)^2$$

for $k = 2, 3, \dots, n$,

- $\phi_k(b) := \arg \min_{\tilde{b}} h_k(\tilde{b}) + \lambda_k |\tilde{b} - b|$.
- $h_k(b) := \frac{1}{2} (x_k - b)^2 + h_{k-1}(\phi_{k-1}(b)) + \lambda_{k-1} |\phi_{k-1}(b) - b|$

Theorem 4.2 (Johnson [2013]). *The function $h_k(b)$ is strictly convex and differentiable. The function $\phi_k(b)$ is piece-wise linear.*

The b and \tilde{b} in the definition of ϕ_k represent the former centroid a_k and the later centroid a_{k+1} respectively. Once \hat{a}_{k+1} is known, by the definition, \hat{a}_k can be expressed as

$$\hat{a}_k = \phi_k(\hat{a}_{k+1}).$$

From the above theorem 4.2, we know $h_k(b)$ is differentiable, hence we define some intermediate notation:

$$g_k(b) := \frac{\partial h_k(b)}{\partial b}, \quad U_k := \arg \min_b h_k(b) - \lambda_k b.$$

It is not straightforward to see but once we know the \hat{a}_{k+1} , the \hat{a}_k can be written in a closed form. Because the former centroid a_k is always equal or smaller than a_{k+1} , in other words $\tilde{b} \leq b$, thus the $|\tilde{b} - b|$ in ϕ_k takes either $b - \tilde{b}$ or 0.

Algorithm 1: DP algorithm

Input: sorted input $x_1 \leq \dots \leq x_n, \lambda$.

Output: $(\hat{a}_1, \dots, \hat{a}_n)$

- 1 Initialize $\lambda_i \leftarrow i(n-i)\lambda$, for $i = 1, \dots, n-1$.
 - 2 **for** $k \leftarrow 1$ **to** $n-1$ **do**
 - 3 Find the U_k
 - 4 **end**
 - 5 Solve \hat{a}_n such that $h_n(\hat{a}_n) = 0$.
 - 6 **for** $k \leftarrow n-1$ **to** 1 **do**
 - 7 $\hat{a}_k = \max(\hat{a}_{k+1}, U_k)$
 - 8 **end**
-

In the case of $\tilde{b} < b$, which corresponds to the case that $\hat{a}_k < \hat{a}_{k+1}$,

$$\begin{aligned} \hat{a}_n &= \arg \min_{\tilde{b}} h_k(\tilde{b}) + \lambda_k (b - \tilde{b}) \\ &= \arg \min_b h_k(\tilde{b}) - \lambda_k \tilde{b} = U_k. \end{aligned}$$

Otherwise $\tilde{b} - b = 0$, which corresponds to the case that $\hat{a}_k = \hat{a}_{k+1}$. By the assumption we already know the \hat{a}_{k+1} , we can simply assign the known \hat{a}_{k+1} to \hat{a}_k . In short, we take the maximum between \hat{a}_{k+1} and U_k and assign it to \hat{a}_k . It is clear that once we find \hat{a}_n and U_1, \dots, U_{n-1} , we can obtain all the centroids by tracking back from $n-1, \dots, 1$. This is summarized in algorithm 1, giving a high-level view of the DP algorithm.

Next, we show how to find U_k for $k = 1, \dots, n-1$. The details are explained in algorithm 2. By KKT condition, U_k satisfies

$$g_k(U_k) - \lambda_k = 0.$$

When $k = 1$, $g_1(U_1) - \lambda_1 = U_1 - x_1 - \lambda_1$ and $U_1 = x_1 + \lambda_1$. For $k = 2, \dots, n-1$, by the definition of $h_k(b)$, we have the derivative of $h_k(b)$ is:

$$g_k(b) = g_{k-1}(b) \mathbf{I}[b \leq U_{k-1}] + \lambda_{k-1} \mathbf{I}[b > U_{k-1}] + (b - x_k),$$

where \mathbf{I} is the indicator function. It is easy to see the function g_k is a piecewise linear function connected by a knot point U_{k-1} : when $b > U_{k-1}$, g_k is a line with the slope 1 and the intercept $\lambda_{k-1} - x_k$; when $b \leq U_{k-1}$, again, it becomes piecewise linear with a new knot point U_{k-2} . Because $g_{k-1}(b)$ includes the $(b - x_{k-1})$ term, so the slope becomes steeper as b becomes smaller. Because g_k is piecewise linear, the key to find the U_k that satisfies $g_k(U_k) = \lambda_k$ is to decide which part of the line is U_k on. To do that, we start to search from the right to left. If the U_k is not on the current line, move left and update the intercept and slope until we find the line where (U_k, λ_k) is. As for \hat{a}_n , it is the same as finding the U_n that satisfying $g_n(U_n) = \lambda_n$ with $\lambda_n = 0$.

In addition, while we search for each U_k , some care to be taken to guarantee it has $\mathcal{O}(n)$ worst case performance, which is the erase step in line 11 of the algorithm 2. In

Algorithm 2: Finding U

Input: $x_1 \leq \dots \leq x_n$, $(\lambda_1, \dots, \lambda_{n-1})$.
Output: (U_1, \dots, U_{n-1})

- 1 // Initialization;
- 2 $U_1 \leftarrow x_1 + \lambda_1$.
- 3 $U^* \leftarrow U_1$, $S^* \leftarrow 1$, $I^* \leftarrow -x_1$.
- 4 **for** $k \leftarrow 2$ **to** $n - 1$ **do**
- 5 $S_k \leftarrow 1$, $I_k \leftarrow -x_k$.
- 6 $\beta \leftarrow (\lambda_k - \lambda_{k-1} - I_k)/S_k$.
- 7 // search from the right side.
- 8 // **.end** denotes the last item of a sequence.
- 9 **while** $U^*.end > \beta$ **do**
- 10 update $S_k \leftarrow S_k + S^*.end$, $I_k \leftarrow I_k + I^*.end$
- 11 erase the last item of U^* , S^* , I^* .
- 12 // suppose the index of $U^*.end$ is U_m ,
- 13 // then update the β as follows:
- 14 $\beta \leftarrow (\lambda_k - \lambda_m - I_k)/S_k$.
- 15 **if** U^* is empty **then**
- 16 break
- 17 **end**
- 18 **end**
- 19 update $U_k \leftarrow \beta$.
- 20 update $U^* \leftarrow (U^*, U_k)$, $S^* \leftarrow (S^*, S_k)$,
- 21 $I^* \leftarrow (I^*, I_k)$.
- 21 **end**

the algorithm 2, the U^* , S^* and I^* can be viewed as three different stacks, each time we enter the inner loop in line 9, we pop the last items of U^* , S^* and I^* out, and after finding the U_k , in line 20 we push the new U_k , S_k and I_k into each stack respectively.

The technical details here is somehow difficult to understand. In order not to interrupt with the flow of the paper, we include an example of $n = 3$ in the supplementary materials, which we believe is helpful in understanding the algorithm.

4.3 C-PAINT ALGORITHM

The DP algorithm is intended for a single tuning parameter λ . However, sometimes it is of our interest to visualize the full clusterpath. Using the proposition result, we first find the λ_{\max} that yields non-trivial solution. In other words, any tuning parameter bigger than λ_{\max} results in one cluster. Next, we proposed the C-PAINT algorithm.

Proposition 4.3 (Radchenko and Mukherjee [2017]). *Given data $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{R}^n$. In the problem 2, the tuning parameter λ_{\max} that yields non-trivial solution is:*

$$\lambda_{\max}(\mathbf{x}) = \max_{j=1, \dots, n-1} \left(\bar{\mathbf{x}} - \frac{1}{j} \sum_{k=1}^j x_k \right) / (n - j)$$

where the $\bar{\mathbf{x}} = \frac{1}{n} \sum_{k=1}^n x_k$.

Algorithm 3: C-PAINT algorithm

Input: Data matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$, λ sequence length K .
Output: $(\hat{\mathbf{A}}_1, \dots, \hat{\mathbf{A}}_K)$, and $\hat{\mathbf{A}}_k = (\hat{\mathbf{a}}_1^k, \dots, \hat{\mathbf{a}}_n^k)^T \in \mathbb{R}^{n \times p}$.

- 1 // find the $\lambda_{\max}(\mathbf{X})$.
- 2 Initialize $\lambda_k \leftarrow \lambda_{\max} \cdot k/K$, $k = 1, \dots, K$.
- 3 **for** $i = 1$ **to** p **do**
- 4 Sort $\mathbf{x}_i \in \mathbb{R}^n$ in a descending order as $\mathbf{x}_{(i)}$ and save the order of the \mathbf{x}_i .
- 5 **for** $k = 1$ **to** K **do**
- 6 // for each dimension i , use the DP algorithm.
- 7 $\hat{\mathbf{a}}_{(i)}^k \leftarrow \text{DP}(\mathbf{x}_{(i)}, \lambda_k)$.
- 8 **end**
- 9 rematch the $\hat{\mathbf{a}}_i^k[\text{order}] \leftarrow \hat{\mathbf{a}}_{(i)}^k$.
- 10 **end**

Similarly, for a given matrix $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_p) \in \mathbb{R}^{n \times p}$, we can obtain the λ_{\max} by taking the maximum value of all dimensions:

$$\lambda_{\max}(\mathbf{X}) = \max_{i=1, \dots, p} \lambda_{\max}(\mathbf{x}_i).$$

This is useful to us that we only need to consider the λ sequence that is smaller than the λ_{\max} in constructing the solution path. In the algorithm 3, we choose a series of tuning parameter λ s as an arithmetic sequence but it is also possible to use a geometric sequence.

4.4 TIME COMPLEXITY

In order to analyse the time complexity, we take a look at how many operations are involved.

In the algorithm 2, U_1 can be founded in $\mathcal{O}(1)$. In finding U_2, \dots, U_{n-1} , line 2-3, line 5-6 and line 19-20 can be calculated in $\mathcal{O}(1)$, and inside the while loop, line 10-16 also takes $\mathcal{O}(1)$, so the key problem is how many times we enter the inner while loop. Every time we enter the inner loop, the last item of the sequence U^* is deleted and never used, and from line 19 we know every U_k will be added once and deleted at most once, thus we can enter the inner loop at most $n - 2$ times, and each time it takes $\mathcal{O}(1)$, so in total the algorithm 2 is $\mathcal{O}(n)$.

Besides, in order to reformulate the problem, first we need to sort (x_1, \dots, x_n) in a ascending order, where the time complexity depends on the choice of the sorting algorithm, we adapt the quick sort algorithm which on average takes $\mathcal{O}(n \log n)$. For a single tuning parameter λ , the DP algorithm takes $\mathcal{O}(n \log n) + \mathcal{O}(n)$. As for the C-PANT algorithm, for each dimension we only need to sort it once to construct the clusterpath, we solve each sub-problem K times with different λ using the DP algorithm, and the time complexity of it is $\mathcal{O}(pnK)$. In total, the C-PAINT takes $\mathcal{O}(pn \log n) + \mathcal{O}(pnK)$.

5 EXPERIMENTAL EVALUATION

In this section, we first provide the run times comparison. The C-PAINT is compared with several representative methods: CARP, FLSA, ADMM, AMA and the FUSION algorithm proposed by Radchenko and Mukherjee [2017]. Results show that our proposed method C-PAINT is significantly faster than other methods in finding clusterpath. Next, the numerical experiments results on both synthetic and real data are provided. As the proposed method is a novel optimization method that yields the exact solution, we focus on showing the recovery of the clusterpath. In the synthetic data example, we generated five clusters with different shapes, the obtained clusterpath shows how each cluster merged along the clusterpath. In the real data examples, we perform C-PAINT and other methods on relatively small datasets and present the run times. We also apply C-PAINT to obtain the clusterpaths on larger datasets which are infeasible for the existing methods.

5.1 IMPLEMENTATION DETAILS

Our proposed DP algorithm and C-PAINT are implemented in Rcpp, which are implemented in the `dpcc` R package. We compare with the CARP function which is implemented in C++ in the `clustRviz` R package, and the tuning parameters are set as recommended values. The FLSA function is in the `flsa` R package, which is implemented in C++. To make a fair comparison, we run the FLSA function without checking the splits based on theorem 3.1. The ADMM and AMA are implemented in the `cvxclustr` R package using R and C. For ADMM and AMA, we set the step size to be $1/n$, and the convergence tolerance to be 10^{-5} . The FUSION algorithm is implemented in R in the `fusionclust` R package. To make a fair comparison, we implemented the code in Rcpp by ourselves and made some modifications to accelerate the algorithm. Because Radchenko and Mukherjee [2017] proposed two similar methods, we only consider the one that successively merges in a bottom-up fashion. As for the modifications, specifically, instead of storing most of the fusion events, we stored the clustering results for only K times, which is equivalent to the length of the λ sequence used in C-PAINT, ADMM and AMA. This modification makes the algorithm much more efficient. Even that, FUSION is still slower than the C-PAINT for large sample size cases.

Our experiments are performed on a MacBook Air with M1 CPU with 8 GB memory. The elapsed times (wall clock times) are taken as the run times.

5.2 TIME COMPARISON

The simulated data consists of 100, 500, 1000, 5000, 10000 and 50000 points in \mathbb{R}^2 from a gaussian mixture model

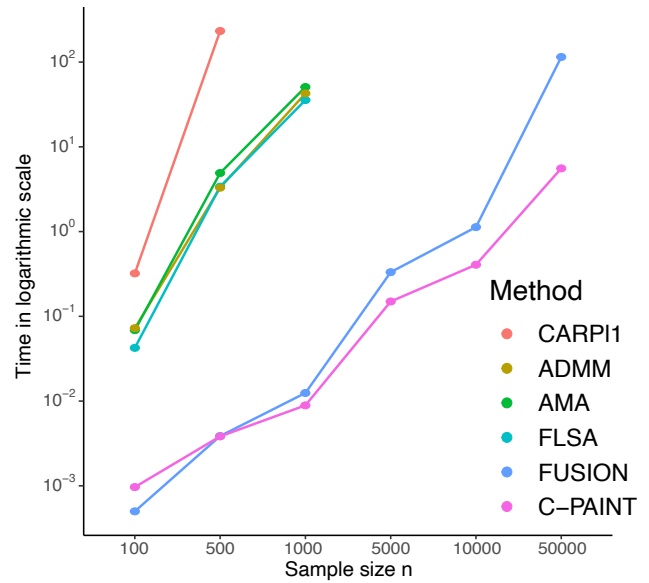


Figure 3: Comparison of the run times in computing clustering solution path in logarithmic scale. Each line represents the run times of different methods.

with three components. The length of the λ sequence is $K = 10$, which means we equally separate the λ sequence to be $(\lambda_{\max}/10, 2\lambda_{\max}/10, \dots, \lambda_{\max})$. All the run times are the means on 30 replications. We let the CARP only run up to 500 and the FLSA, ADMM and AMA only run up to 1000 points because it took much more time for bigger sample sizes. However, for the FUSION algorithm and the proposed method, we run over large data sets of 5000, 10000 and 50000. Although for L_1 case, each sub-problems can be solved in parallel for C-PAINT, FLSA and FUSION algorithm, we do not pursue it here.

The time comparison results are shown in the Figure 3. In the figure 3, the x-axis shows the sample size n , and the y-axis shows the run times in second in logarithmic scale.

We notice for the identical weights and L_1 penalty setting, CARP is slower than other methods when the length K is small. FLSA, ADMM and AMA show quite close performance, and is generally slower than C-PAINT and FUSION algorithm. FUSION algorithm is fast but unable to create a clusterpath visualization because it does not estimate the centroids with a given tuning parameter λ . In terms of visualize the clusterpath and do clustering, C-PAINT is efficient and fast. In particular, C-PAINT can find the full solution path of 10^7 samples in \mathbb{R}^2 within two minutes. Moreover, simulation shows the run times of C-PAINT grow linearly, which coincides with the time complexity analysis in Section 4. We can see C-PAINT is generally faster than FUSION. The possible reason is that even though we save it from storing most of the events, it still needs to go through every fusion events, which makes it less efficient as the sample size grows. Meanwhile, the proposed method

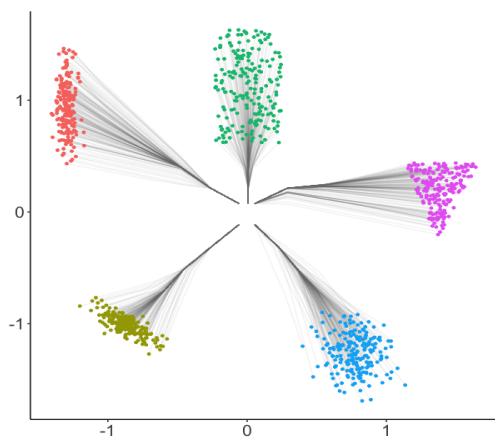


Figure 4: Visualization of the clusterpath generated with a λ sequence with length $K = 10$. The colors show the clustering results with the biggest tuning parameter of the lambda sequence. The threshold γ are set to be 10^{-6} .

performs the DP algorithm independently with a single λ . In conclusion, in terms of finding clusterpath, C-PAINT outperforms the CARP, FLSA, ADMM and AMA and is capable of solving large scale problems. The details are in the Appendix.

5.3 SYNTHETIC DATA

We generate the synthetic data into five clusters with different shapes, each includes 200 data points. For a better interpretability of the clustering results, we set a small threshold $\gamma = 10^{-6}$ that for all the estimated centroids within the euclidean distance of γ , we put them into a same cluster.

In the Figure 4, both clustering results and clusterpath are presented in the same plots. The colors represent clusters obtained by the DP algorithm with a certain parameter λ . Instead of drawing the full clusterpath, we stop it halfway before all the points collapse into one final cluster. By distinguishing the merged centers, the convex clustering successfully separated different clusters.

5.4 SMALL DATASET

We use two small real datasets to investigate the performance of our proposed algorithm and other methods.

- **Lymphoma** [Alizadeh et al., 2000] dataset includes 62 samples categorized into three lymphoma types.
- **Gene expression** [Weinstein et al., 2013] dataset includes the gene expression features of 801 samples with four different types of tumor as labels.

To better visualize the high-dimensional datasets, we first use the UMAP [McInnes et al., 2018] to reduce the dimen-

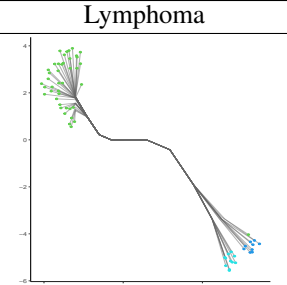
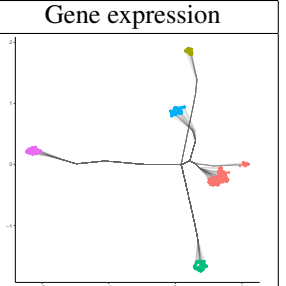
Lymphoma	Gene expression
	
Sample size : $n = 62$	$n = 801$
Run time (second)	
CARP: 6.6×10^{-2}	223
ADMM: 1.1×10^{-2}	8.6
C-PAINT: 4.7×10^{-4}	0.012

Table 1: Visualization of the clusterpath generated with a λ sequence with length $K = 10$. The colors show the original labels of the samples. Run times are the means over 30 replications.

sions into two. We use the umap function in the `uwot` R package. Next, we perform the C-PAINT on the projected coordinates. The colors show the original labels of the data.

Both the full clusterpaths and run times are reported in the table 1. We only report the ADMM since the ADMM, AMA, FLSA have similar run times. From the result, we can see C-PAINT is much faster than other methods.

5.5 LARGE DATASET

Now we use relatively large datasets to investigate the performance of the proposed algorithm. In particular, we use:

- **Frey faces** dataset includes 1965 images of Brendan Frey’s face, taken from sequential frames of a small video. This is included in the `snedata` R package.
- **RNA sequence** [Stuart et al., 2019] multi-dataset includes 5683 cells consisting of 11 cell types and differentially expressed genes as their features.
- **Anuran (frog) calls** [Han and Zhang, 2016] dataset includes the extracted features from 7195 frog calls records, and each frog has family, genus and species labels, among which we choose the family.
- **Fashion-MNIST** [Xiao et al., 2017] dataset consists of 70000 grayscale images of items such as T-shirt, Trouser and Bag.
- **Kuzushiji-MNIST** [Clanuwat et al., 2018] dataset consists of 70000 grayscale images of hiragana characters in Japanese.

Here we focus on showing the recovery of the clusterpaths on large datasets. The table 2 shows the results of the first four datasets, both the sample sizes and the run times of

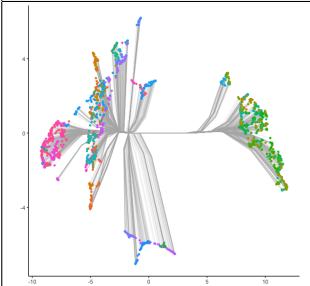
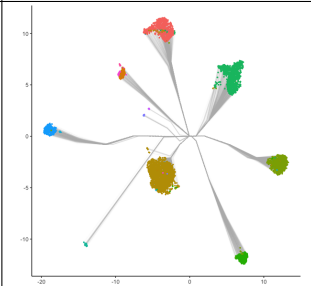
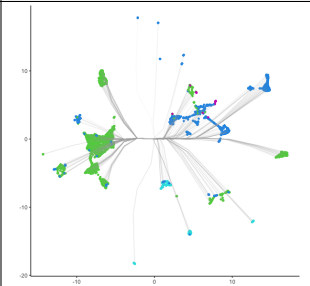
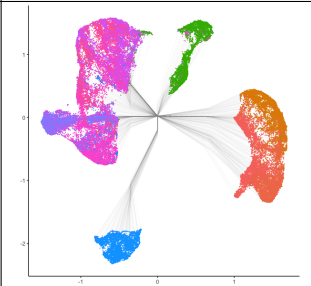
Frey faces	RNA sequence	Anuran (frog) calls	Fashion-MNIST
			
Sample size : $n = 1965$	$n = 5683$	$n = 7195$	$n = 70000$
Run time (s) : 8.4×10^{-3}	2.6×10^{-2}	5.6×10^{-2}	15.8

Table 2: Visualization of the clusterpaths of real datasets. The clusterpaths are drawn by C-PAINT using the coordinates obtained by UMAP. The length of the λ sequence is set to be $K = 5$. The run times of each real datasets are the means over 10 replications.

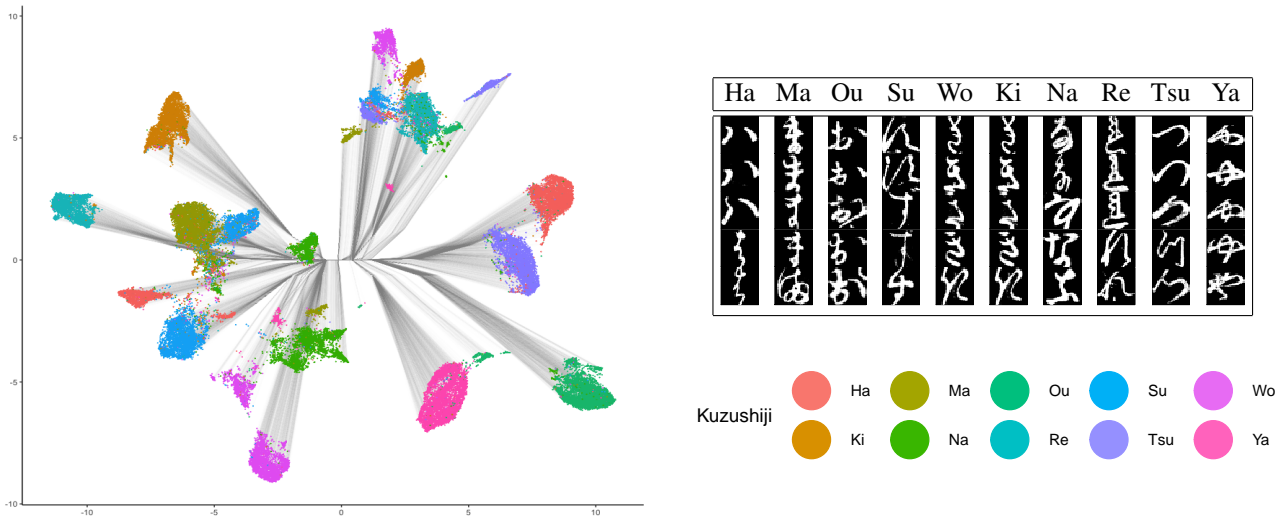


Figure 5: Visualization of the clusterpath of kuzushiji-MNIST is shown on the left. Some representative variants of kuzushiji and the legend are shown on the right. The length of the λ sequence is set to be $K = 5$. The mean run time is 11.578s over ten replications.

the C-PAINT are presented. Next, We choose the kuzushiji dataset to explain in further details.

Kuzushiji-MNIST is a drop-in replacement for the MNIST dataset, consisting of ten rows of Japanese Hiragana. Different from the ordinary Hiragana used in Japanese nowadays, the kuzushiji came from ancient Chinese characters variants thus each has several variants. For example, we can see in the Figure 5 that except for the Hiragana Ha, others has two or more variants that looks quite unlike. To be more specific, we can see on the right side of the clusterpath, Ha and Tsu merge along the clusterpath at an early stage because the projection of the images share great similarity.

6 CONCLUSIONS

We proposed a novel algorithm for L_1 convex clustering. To the best of our knowledge, it is the first time that dynamic programming is applied to the convex clustering problem. We reformulated the sub-problems for each dimension into weighted one-dimensional fused lasso problems, which can be solved with a dynamic programming algorithm.

In order to visualize the clusterpath, we proposed the C-PAINT based on the DP algorithm. The time complexity of C-PAINT is $\mathcal{O}(pn \log n) + \mathcal{O}(pnK)$, but in practice it grows linearly with respect to sample size n and dimension p . The proposed algorithm is highly efficient and outperforms the existing algorithms.

For L_1 convex clustering with identical weights, the simulation results show our proposed method overcomes the

computational bottleneck of the convex clustering, making it possible to recover the full clusterpath for large datasets. Our methods are implemented in the R package `dpcc`, which is also available at <https://github.com/bingyuan-zhang/dpcc>.

Author Contributions

Bingyuan Zhang and Yoshikazu Terada were the main contributing authors to this work.

Acknowledgements

This research was supported in part by JSPS KAKENHI Grant (20K19756 to YT).

References

Ash A Alizadeh, Michael B Eisen, R Eric Davis, Chi Ma, Izidore S Lossos, Andreas Rosenwald, Jennifer C Boldrick, Hajeer Sabet, Truc Tran, Xin Yu, et al. Distinct types of diffuse large b-cell lymphoma identified by gene expression profiling. *Nature*, 403(6769):503–511, 2000.

Eric C Chi and Kenneth Lange. Splitting methods for convex clustering. *Journal of Computational and Graphical Statistics*, 24(4):994–1013, 2015.

Eric C Chi, Genevera I Allen, and Richard G Baraniuk. Convex biclustering. *Biometrics*, 73(1):10–19, 2017.

Julien Chiquet, Pierre Gutierrez, and Guillem Rigaill. Fast tree inference with weighted fusion penalties. *Journal of Computational and Graphical Statistics*, 26(1):205–216, 2017.

Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature, 2018.

Lei Han and Yu Zhang. Reduction techniques for graph-based convex clustering. In *AAAI*, pages 1645–1651, 2016.

Toby Dylan Hocking, Armand Joulin, Francis Bach, and Jean-Philippe Vert. Clusterpath an algorithm for clustering using convex fusion penalties. In *28th international conference on machine learning*, page 1, 2011.

Holger Hoeffling. A path algorithm for the fused lasso signal approximator. *Journal of Computational and Graphical Statistics*, 19(4):984–1006, 2010.

Nicholas A Johnson. A dynamic programming algorithm for the fused lasso and l₀-segmentation. *Journal of Computational and Graphical Statistics*, 22(2):246–260, 2013.

Fredrik Lindsten, Henrik Ohlsson, and Lennart Ljung. Clustering using sum-of-norms regularization: With application to particle filter output computation. In *2011 IEEE Statistical Signal Processing Workshop (SSP)*, pages 201–204. IEEE, 2011.

Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

Kristiaan Pelckmans, Joseph De Brabanter, Johan AK Suykens, and B De Moor. Convex clustering shrinkage. In *PASCAL Workshop on Statistics and Optimization of Clustering Workshop*, 2005.

Peter Radchenko and Gourab Mukherjee. Convex clustering via l₁ fusion penalization. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 79(5):1527–1546, 2017.

Tim Stuart, Andrew Butler, Paul Hoffman, Christoph Hafemeister, Efthymia Papalexi, William M Mauck III, Yuhan Hao, Marlon Stoeckius, Peter Smibert, and Rahul Satija. Comprehensive integration of single-cell data. *Cell*, 177(7):1888–1902, 2019.

Kean Ming Tan and Daniela Witten. Statistical properties of convex clustering. *Electronic journal of statistics*, 9(2):2324, 2015.

Robert Tibshirani, Michael Saunders, Saharon Rosset, Ji Zhu, and Keith Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):91–108, 2005.

Binhuan Wang, Yilong Zhang, Will Wei Sun, and Yixin Fang. Sparse convex clustering. *Journal of Computational and Graphical Statistics*, 27(2):393–403, 2018.

Q. Wang, P. Gong, S. Chang, T. S. Huang, and J. Zhou. Robust convex clustering analysis. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 1263–1268, 2016. doi: 10.1109/ICDM.2016.0170.

John N Weinstein, Eric A Collisson, Gordon B Mills, Kenna R Mills Shaw, Brad A Ozenberger, Kyle Ellrott, Ilya Shmulevich, Chris Sander, and Joshua M Stuart. The cancer genome atlas pan-cancer analysis project. *Nature genetics*, 45(10):1113–1120, 2013.

Michael Weylandt, John Nagorski, and Genevera I Allen. Dynamic visualization and fast computation for convex clustering via algorithmic regularization. *Journal of Computational and Graphical Statistics*, 29(1):87–96, 2020.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Yancheng Yuan, Defeng Sun, and Kim-Chuan Toh. An efficient semismooth newton based algorithm for convex clustering. In *International Conference on Machine Learning*, pages 5718–5726. PMLR, 2018.

Changbo Zhu, Huan Xu, Chenlei Leng, and Shuicheng Yan. Convex optimization procedure for clustering: Theoretical revisit. *Advances in Neural Information Processing Systems*, 27:1619–1627, 2014.