

Title	Towards Advanced Unsupervised Learning
Author(s)	Zhang, Bingyuan
Citation	大阪大学, 2023, 博士論文
Version Type	VoR
URL	https://doi.org/10.18910/92903
rights	
Note	

Osaka University Knowledge Archive : OUKA

<https://ir.library.osaka-u.ac.jp/>

Osaka University

Towards Advanced Unsupervised Learning

BINGYUAN ZHANG

JUNE 2023

Towards Advanced Unsupervised Learning

A dissertation submitted to
THE GRADUATE SCHOOL OF ENGINEERING SCIENCE
OSAKA UNIVERSITY
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY IN ENGINEERING

By

BINGYUAN ZHANG

JUNE 2023

Abstract

Unsupervised learning is an important category in machine learning algorithms. Unsupervised learning provides a powerful tool for exploring unlabeled datasets and extracting useful information. Unlike supervised learning, unsupervised learning is trained without any explicit supervision. This makes unsupervised learning an attractive option since human-labeled data can be expensive to obtain, while unlabeled data are easily collected. On the other hand, large-scale and high-dimensional datasets are common in fields such as computer vision, natural language processing, and biology. Therefore, there is a growing demand for efficient and high-performance unsupervised learning algorithms that can handle the complexity of large-scale and high-dimensional datasets.

In the first part, we study convex clustering. Clustering is a popular unsupervised learning technique. There are many famous clustering methods, such as k -means and hierarchical clustering methods. Convex clustering is a modern clustering framework with both features of k -means and hierarchical clustering. We proposed a highly efficient L_1 -convex clustering method that is capable of visualizing the clusterpath on large datasets. Results show the proposed method is significantly more efficient than existing methods in finding the optimal solution.

In the second part, we study the hypothesis test on conditional independence. Conditional Independence (CI) test is a fundamental problem in statistics. It aims to determine whether variables X and Y are conditionally independent given another variable Z fixed. However, when the conditioning set Z is continuous and high-dimensional, it becomes a challenging problem. We proposed a robust CI test that outperforms existing tests against the growth of the dimension of Z . Results show the proposed method provides a more accurate and reliable way of determining conditional independence.

Table of Contents

1	Introduction	1
1.1	Motivation	1
1.2	Thesis organization and contribution	2
2	Background	4
2.1	Background on convex clustering	4
2.1.1	Lasso regression	4
2.1.2	Fused Lasso	5
2.1.3	Optimization algorithms	6
2.2	Background on conditional independence	8
2.2.1	About kernel methods	8
2.2.2	Independence	12
2.2.3	Conditional independence	13
3	Convex clustering	16
3.1	Introduction	16
3.2	Related work	20
3.3	Preliminaries	21
3.4	Proposed method	22
3.4.1	Idea	22
3.4.2	DP algorithm	24
3.4.3	C-PAINT algorithm	28
3.4.4	Time complexity	29
3.5	Experimental evaluation	31
3.5.1	Implementation details	31
3.5.2	Time comparison	32

3.5.3	Synthetic dataset	34
3.5.4	Small real dataset	35
3.5.5	Large real dataset	36
3.6	Summary	38
4	Conditional independence test	41
4.1	Introduction	41
4.2	Related work	44
4.3	Background on kernel methods	45
4.4	Proposed method	47
4.4.1	Local bootstrap	49
4.4.2	Effect of M	52
4.4.3	Complexity analysis	52
4.5	Experiments	53
4.5.1	Hyperparameters setting	54
4.5.2	When Z is low-dimensional	55
4.5.3	When Z is high-dimensional	56
4.5.4	When d_Z changes	57
4.5.5	Effect on cluster number M	58
4.6	Summary	59
5	Conclusions and future work	60
5.1	Conclusions	60
5.2	Discussion	61
6	Appendix	62
6.1	An example $N = 3$	62
6.2	Simulation details	64

List of publications	66
Acknowledgements	67
References	68

List of Figures

3.1	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.	18
3.2	Replacement of the penalty graph. The edges are the absolute values of the differences between nodes. The total sums of the edges' lengths are identical for the two graphs, which inspires us to reformulate the penalty term.	23
3.3	Comparison of the runtimes in computing clustering solution path in logarithmic scale. Each line represents the runtimes of different methods.	33
3.4	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}	34
3.5	Some representative variants of kuzushiji and the legend are shown on the top. The middle figure show the visualization of the clusterpath of kuzushiji-MNIST. The length of the λ sequence is set to be $K = 5$. The mean runtime is 11.578 seconds over ten replications.	40
4.1	An ideal permutation in CI.	50
4.2	Simulation results on linear model 1 ($d_Z = 1$). The significant level is $\alpha = 0.05$. Type-I error rates, Type-II error rates are reported.	55
4.3	Simulation results on non-linear model 2 ($d_Z = 1$). The significant level is $\alpha = 0.05$. Type-I error rates, Type-II error rates and mean runtimes are reported.	55

4.4	Simulation results on linear model 1 ($d_Z = 10$). The significant level $\alpha = 0.05$. Type-I error rates, Type-II error rates are reported.	56
4.5	Simulation results on non-linear model 2 ($d_Z = 10$). The significant level $\alpha = 0.05$. Type-I error rates, Type-II error rates and mean runtimes are reported.	57
4.6	Simulation results on different dimensions of Z ($d_Z = 1, 5, 10, 15, 20$). The sample size $n = 400$ is fixed. The significant level $\alpha = 0.05$. Type-I error rates, Type-II error rates and mean runtimes are reported.	58
4.7	Simulation results on a different cluster number M . The sample size $n = 400$ is fixed. Results on different dimensionality of Z are reported ($d_Z = 1$, red line; $d_Z = 10$, blue line). The significant level $\alpha = 0.05$. Type-I error rates, Type-II error rates and mean runtimes are reported.	59
6.1	An image of $g_2(b)$: the solid line is an example of $g_2(b)$ with a branch point U_1 . The search starts to find the β first and if β does not qualify $\beta > U_1$, then we go to find the β_{new} .	64

List of Tables

2.1	Optimization methods for the fused Lasso problems.	7
3.1	Visualization of the clusterpath generated with a λ sequence with length $K = 10$. The colors show the original labels of the samples. runtimes are the means over 30 replications.	36
3.2	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 runtimes of each real dataset are the means over 10 replications.	37
6.1	Run times Comparison. The means and standard errors of each method over 30 replications are reported. Here * means we cannot obtain the solutions within a reasonable time.	65

1 | Introduction

1.1 Motivation

The increasing availability of large-scale and high-dimensional datasets has created a need for efficient and better-performed unsupervised learning methods. This thesis addresses the following two tasks in unsupervised learning. We describe current challenges and clarify motivations for developing better algorithms.

Convex clustering is a modern unsupervised learning technique that has recently gained popularity. The convex clustering algorithms involve two steps: first, solving a fused Lasso problem and then determining the clusters based on the estimated parameters of the fused Lasso problem. The major bottleneck of convex clustering lies in the difficulty of solving the fused Lasso problem. Although significant progress has been made in developing fast optimization algorithms, applying convex clustering on large-scale datasets is still challenging. This limits the practical application of convex clustering to small datasets. To overcome this challenge, we aim to develop a more efficient convex clustering optimization procedure to make convex clustering applicable to large-scale datasets.

Conditional independence test aims to find whether variables X, Y are independent given another variable Z . Independence test and conditional independence test play a central role in constrained-based causal discovery. CI tests are applied to identify the presence or absence of causal links between variables. In that case, usually, the conditioning Z is high-dimensional. However, current CI tests have difficulty dealing with high-dimensional conditioning Z . To address this problem, we aim to develop a robust and efficient nonparametric CI test that can handle high-dimensional conditioning Z .

1.2 Thesis organization and contribution

The thesis is organized as follows: In Chapter 1, we briefly introduce the motivation of this thesis and present a summary of contributions. In Chapter 2, we provide related background information that is crucial for understanding the following contents. For convex clustering, we introduce sparse estimation, particularly the fused Lasso problem, because the fused Lasso problem is a critical component in the loss of convex clustering methods. For conditional independence test, we provide a background on kernel methods and the characterizations of independence and conditional independence. In Chapter 3, we propose a method called C-PAINT for L_1 convex clustering with identical weight, an important special case for L_1 convex clustering. In particular,

Contribution of Chapter 3

- We reveal that the sub-problem of L_1 convex clustering with identical weight can be reformulated into a weighted 1d-fused Lasso problem.
- We propose a refined DP algorithm to solve the reformulated weighted 1d-fused Lasso problem.
- We propose C-PAINT to construct a full clusterpath. In practice, we show that C-PAINT is highly efficient and possible for large datasets.

In Chapter 4, we propose a novel procedure to perform a non-parametric CI test. In particular,

Contribution of Chapter 4

- We design a procedure to find a novel test statistic. We first subdivide the conditioning set Z into several local clusters, measure the unconditional independence of (X, Y) in each cluster, and consider the sum of local un-

conditional independence measures as the test statistic.

- We propose a local bootstrap to sample from the CI case $H_0 : X \perp\!\!\!\perp Y \mid Z$. The local bootstrap method works well with the proposed test statistic and can be applied to other CI tests.

In Chapter 5, we summarize the proposed methods and provide a comprehensive discussion of their strengths and limitations. We provide a discussion about possible directions for the follow-up work.

2 | Background

2.1 Background on convex clustering

Clustering methods play a vital role in the field of unsupervised learning. Clustering methods are powerful tools in exploratory data analysis and have a wide range of applications in various fields. Numerous methods have been developed over the years; among them, convex clustering is a modern framework designed to perform clustering by minimizing a convex loss function. The convex clustering's loss function is closely related to the sparse models, specifically to the fused Lasso problem. Therefore, we provide some helpful background information on sparse estimation and fused Lasso.

2.1.1 Lasso regression

We consider a linear regression model as follows:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \quad (2.1)$$

where the $\mathbf{y} \in \mathbb{R}^n$ is the response vector and $\mathbf{X} \in \mathbb{R}^{n \times p}$ is a design matrix of the predictor variables, and $\boldsymbol{\beta} \in \mathbb{R}^p$ is a p -dimensional coefficient vector. $\boldsymbol{\varepsilon} \in \mathbb{R}^n$ is the error vector that each element is assumed to be i.i.d. with zero mean. The linear model is a simple framework and provides predictive performance in many practical situations. However, in scenarios where the predictor variables are high-dimensional and only a few true coefficients are non-zero, the least squares estimator may not be a feasible option. To address the problem, penalized models have become of interest for applications that aim to promote the recovery of the true coefficients $\boldsymbol{\beta}$ and learn a sparse representation.

Penalized models have been a well-studied subject, and among them, the Least Absolute Shrinkage and Selection Operator (Lasso) [1] is a famous model. Lasso adds a penalty term on β with a tuning parameter λ to the squared error loss function. In the linear regression setting, Lasso minimizes the loss:

$$\frac{1}{2} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda \sum_{i=1}^p |\beta_i|. \quad (2.2)$$

The penalty terms of absolute value $|\cdot|$ promote sparsity in the coefficient vector. As the tuning parameter λ increases, the estimated coefficients are progressively shrunk toward zero, resulting in a model with fewer non-zero estimated elements in the parameter vector $\hat{\beta}$.

2.1.2 Fused Lasso

The fused Lasso [2] is an extension of the Lasso that was initially proposed to encourage the smoothness and local constancy of coefficients. In the linear regression setting, the fused Lasso minimizes the following loss:

$$\frac{1}{2} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda \sum_{i=1}^{n-1} |\beta_i - \beta_{i+1}|. \quad (2.3)$$

The penalty terms are the absolute value of the difference between adjacent coefficients, which encourages the similarities between adjacent coefficients.

A more widely applied use of the fused Lasso is to decrease noise and restore the original signal. The one dimension Fused Lasso Signal Approximator (FLSA) [3] considers the case with $\mathbf{X} = \mathbf{I}$ and the loss function becomes:

$$\frac{1}{2} \|\mathbf{y} - \beta\|_2^2 + \lambda \sum_{i=1}^{n-1} |\beta_i - \beta_{i+1}|. \quad (2.4)$$

In the above loss function, the parameter vector β is used to approximate the true signal from noisy input \mathbf{y} . We observe that additional penalty terms are placed on the difference between neighboring coefficients. Such penalty terms are applied to smooth the signals if the input data has a natural order. However, we may consider the penalty terms of different connected pairs beyond the local neighbors in a more general form. We have the following loss function of the generalized fused Lasso [4]:

$$\frac{1}{2} \|\mathbf{y} - \beta\|_2^2 + \lambda \sum_{(i,j) \in E} |\beta_i - \beta_{i+1}|. \quad (2.5)$$

The E is an edge set of a general graph, and the penalty terms can be placed on any pairs between coefficients. If the graph is a chain graph, then the problem (2.5) degenerates to the problem (2.4). It is worth noting that in addition to L_1 norm penalty, replacing it with L_2 norm penalty in the FLSA framework can also produce good results. The main difference is that the L_2 penalty produces smoother solutions, while the L_1 penalty promotes sparsity. Therefore, the choice between L_1 and L_2 penalty should depend on the specific application and the desired properties of the resulting solution. Additionally, joint-learning algorithms such as [5, 6] also use the fusion penalties of L_1 or L_2 norms to encourage similarities in parameters.

2.1.3 Optimization algorithms

The fused Lasso problem shares a key property with Lasso, that both of their loss functions (2.2) and (2.5) are convex [2]. The convex formulation ensures the uniqueness of the optimal solution for fused Lasso. However, solving the fused Lasso problem can be a challenging task. Several algorithms have been developed for the fused Lasso. Some methods are designed for the chain-structured fused Lasso problem (2.4) while others are for the general fused Lasso problem (2.5).

We summarize some representative methods in the following table.

Method	Application
CVX (general solver for convex problem)	(2.5)
Component-wise algorithm [7]	(2.5)
Path algorithm [3]	(2.5)
DFS-fused Lasso [8]	(2.5)
general FLSA approximator [3]	(2.5)
1d-FLSA approximator [3]	(2.4)
Dynamic Programming [9]	(2.4)

Table 2.1: Optimization methods for the fused Lasso problems.

For the chain-structured fused Lasso problem (2.4), the dynamic programming (DP) method proposed by Johnson [9] is so far the most efficient optimization method. Unlike some general solvers for the convex optimization problems (e.g., alternating direction method of multipliers (ADMM) [10]), the dynamic programming method does not require iterative updating parameters. The dynamic programming method takes only $\mathcal{O}(n)$ -operations to obtain the exact solution. There are also some variants of the DP method [8, 11], but the structure of weight is limited to chain and tree graphs [9, 12, 11].

In Chapter 3, we demonstrate the application of fusion penalty in clustering, leading to a framework known as convex clustering. Similar to the fused Lasso formulation, convex clustering faces optimization challenges. Various efforts have been made to accelerate the optimization procedure. It is worth mentioning that the optimization methods proposed in the context of convex clustering are not listed here. These methods will be introduced in Chapter 3.2.

2.2 Background on conditional independence

Independence and conditional independence are two fundamental concepts in statistics. Given observations of variables, reliable independence and conditional independence testing methods are essential in constrained-based causal discovery [13, 14]. On the other hand, kernel-based method is an important category in machine learning methods and is widely applied in independence and conditional independence testing. First, we include a gentle introduction to build a basic understanding of kernel. Next, we provide some background information on the definitions and characterizations of unconditional independence and conditional independence, which are closely related.

2.2.1 About kernel methods

This subsection introduces notation, definitions, and basic propositions about kernel methods. See [15, 16, 17] for further details. Throughout the thesis, we consider the positive-definite kernels defined below.

Definition 2.2.1 (Positive-definite Kernel) *Let \mathcal{X} be a nonempty set. Suppose a function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is symmetric, i.e. $k(x_1, x_2) = k(x_2, x_1), \forall x_1, x_2 \in \mathcal{X}$. k is said to be positive definite if $\forall (x_1, \dots, x_n) \in \mathcal{X}^n (n \geq 1)$, the gram matrix*

$$\begin{pmatrix} k(x_1, x_1) & \dots & k(x_1, x_n) \\ \vdots & & \vdots \\ k(x_n, x_1) & \dots & k(x_n, x_n) \end{pmatrix} \in \mathbb{R}^{n \times n} \quad (2.6)$$

is a positive semi-definite matrix.

Proposition 1 *Let k_1, k_2, \dots be positive-definite kernels, then the following functions are positive-definite kernels.*

-
1. $\alpha k_1 + \beta k_2$, $\alpha, \beta \geq 0$.
 2. $k_1 \times k_2$.
 3. $k(x, y) = f(x)f(y)$, where $f(x) \in \mathbb{R}$, $x \in \mathcal{X}$.

The following binary functions are some well-known positive-definite kernels:

Gaussian kernel: $k(x, y) = \exp(-\frac{1}{\sigma} \|x - y\|_2^2)$, $\sigma > 0$.

Laplacian kernel: $k(x, y) = \exp(\alpha \|x - y\|_1)$, $\alpha > 0$.

Polynomial kernel: $k(x, y) = \exp(x^T y + \alpha)^d$, $\alpha \geq 0$, d : degree parameter.

Among them, Gaussian kernel is the most widely applied kernel [18, 17, 19]. Based on the above proposition, the product of Gaussian kernels is also a positive-definite kernel. In the later chapters, we will use Gaussian kernel as a default choice, if not specified.

Definition 2.2.2 (Hilbert Space) *A vector space \mathcal{H} with an inner product defined is called a Hilbert space if it is also a complete metric space with respect to the distance function induced by the inner product.*

Definition 2.2.3 (Reproducing Kernel) *Let \mathcal{H} be a Hilbert space which consists of functions $f : \mathcal{X} \rightarrow \mathbb{R}$. A kernel is called a reproducing kernel if the following two conditions are satisfied:*

- $\forall x \in \mathcal{X}, k(x, \cdot) \in \mathcal{H}$.
- $\forall f \in \mathcal{H}, \forall x \in \mathcal{X}, f(x) = \langle f, k(x, \cdot) \rangle_{\mathcal{H}}$.

\mathcal{H} is called a Reproducing Kernel Hilbert Space (RKHS) with a reproducing kernel k whose span is dense in \mathcal{H} . A positive-definite kernel k can be used to define

a feature map $\Psi : \mathcal{X} \rightarrow \mathcal{H}$, such that $\Psi(x) := k(x, \cdot)$ and

$$k(x_1, x_2) = \langle \Psi(x_1), \Psi(x_2) \rangle_{\mathcal{H}}$$

for $x_1, x_2 \in \mathcal{X}$. The $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ is the inner product of the Hilbert space \mathcal{H} . Such an \mathcal{H} is a reproducing kernel Hilbert space (RKHS) for the kernel k , denoted as \mathcal{H}_k .

Next, we define the following mapping of a measure to an RKHS.

Definition 2.2.4 (Kernel Embedding) *Let \mathcal{P} be a set of measures. The kernel embedding of the measure μ into the RKHS \mathcal{H}_k is the map $m_k : \mathcal{P} \rightarrow \mathcal{H}_k$ defined by*

$$\mathcal{P} \ni \mu \mapsto m_k(\mu) := \int k(\cdot, x) d\mu(x) \in \mathcal{H}_k,$$

From the above definition, a direct consequence is

$$\int f(x) d\mu(x) = \langle f, m_k(\mu) \rangle_{\mathcal{H}_k}, \forall f \in \mathcal{H}_k.$$

A kernel is said to be characteristic if the above mapping is injective; in other words, $m_k(\mu)$ is uniquely embedded in \mathcal{H} . For example, Gaussian kernel is known to be a characteristic kernel [15, 16]. Such a mapping is used in comparison to different distributions. Next, we introduce a distance measure defined between two probability distributions.

Definition 2.2.5 (Maximum Mean Discrepancy (MMD)) *The Maximum Mean Discrepancy (MMD) between $P, Q \in \mathcal{P}$ is defined as*

$$MMD(P, Q) := \|m_k(P) - m_k(Q)\|_{\mathcal{H}_k}.$$

By definition, it is easy to notice that MMD takes non-negative values. In partic-

ular, for characteristic kernels, the $\text{MMD}(P, Q)$ becomes zero if and only if the measures P, Q coincide [19]. The squared MMD has an alternative expression as follows:

$$\begin{aligned}\text{MMD}^2(P, Q) &= \|m_k(P) - m_k(Q)\|_{\mathcal{H}_k}^2 \\ &= \mathbb{E}_{X, X'}[k(X, X')] + \mathbb{E}_{Y, Y'}[k(Y, Y')] - 2\mathbb{E}_{X, Y}[k(X, Y)]\end{aligned}$$

And an unbiased estimator of squared MMD [19] is given by

$$\begin{aligned}\widehat{\text{MMD}}^2 &= \frac{1}{m(m-1)} \sum_{j=1}^m \sum_{i \neq j} k(x_i, x_j) + \frac{1}{n(n-1)} \sum_{j=1}^n \sum_{i \neq j} k(y_i, y_j) \\ &\quad - \frac{2}{mn} \sum_{i=1}^m \sum_{j=1}^n k(x_i, y_j)\end{aligned}$$

Finally, we consider an unconditional dependence measure for variables X and Y . Let $k_{\mathcal{X}}$ and $k_{\mathcal{Y}}$ be kernels on \mathcal{X} and \mathcal{Y} , and $\mathcal{H}_{k_{\mathcal{X}}}$ and $\mathcal{H}_{k_{\mathcal{Y}}}$ be the corresponding RKHSs. Gretton et al. [17] defined the Hilbert-Schmidt independence criterion (HSIC), which can be viewed as the squared MMD between P_{XY} and the product $P_X P_Y$ of the marginalized measures P_X, P_Y . HSIC is a state-of-the-art dependence measure that suits both continuous and discrete variables.

Definition 2.2.6 (Hilbert-Schmidt independence criterion (HSIC))

$$\begin{aligned}\text{HSIC}(X, Y) &:= \|m_k - m_{k_{\mathcal{X}}} m_{k_{\mathcal{Y}}}\|_{\mathcal{H}}^2 \\ &= \|\mathbb{E}_{X, Y}[k_{\mathcal{X}}(X, \cdot) k_{\mathcal{Y}}(Y, \cdot)] - \mathbb{E}_X[k_{\mathcal{X}}(X, \cdot)] \mathbb{E}_Y[k_{\mathcal{Y}}(Y, \cdot)]\|_{\mathcal{H}}^2,\end{aligned}$$

where the \mathcal{H} is the corresponding RKHS of the kernel $k := k_{\mathcal{X}} k_{\mathcal{Y}}$ defined by

$$k((x, y), (x', y')) = k_{\mathcal{X}}(x, x') k_{\mathcal{Y}}(y, y')$$

for $(x, y), (x', y') \in \mathcal{X} \times \mathcal{Y}$.

For a characteristic kernel, the HSIC(X, Y) is zero if and only if $P_{XY} = P_X P_Y$.

2.2.2 Independence

In this subsection, we introduce the definition of independence and some measures for dependence between two variables X, Y .

Definition Given random variables X, Y and we assume the joint probability density p_{XY} and p_X, p_Y exist. X, Y are independent, denoted as $X \perp\!\!\!\perp Y$, when

$$p_{XY} = p_X p_Y$$

Based on finite observation values $\{(x_i, y_i)\}_{i=1}^n$ for variables X, Y , we may consider different dependence measures between X, Y . For example,

Spearman's ρ :

$$\rho = \frac{\text{cov}(r_x, r_y)}{\sigma_{r_x} \sigma_{r_y}}.$$

where the r_x, r_y are the converted rank variables of X, Y . We may convert $\{(x_i, y_i)\}_{i=1}^n$ to $r_x(x_i), r_y(y_i), \forall i = 1, \dots, n$ as their ranks. Let the $d_i := r_x(x_i) - r_y(y_i)$ be the difference between the ranks of the pair (x_i, y_i) , we have

$$\rho = 1 - \frac{6 \sum_{i=1}^n d_i^2}{n(n^2 - 1)}.$$

Kendall's τ :

$$\tau = \frac{1}{n^2} \sum_{i \neq j}^n \text{sign}(x_i - x_j) \text{sign}(y_i - y_j).$$

where $\text{sign}(a) = \mathbf{1}[a > 0]$ takes 1 if $a > 0$ or 0 otherwise. The ties of (x_i, x_j) and (y_i, y_j) pairs are overlooked for simplicity. Both Spearman's ρ and Kendall's

τ are rank correlation statistics. Beyond that category, we may also use the distance between the joint distribution P_{XY} and the marginal distributions P_X and P_Y as dependence measures. We show an information-based measure (Mutual Information) and a kernel-based measure (HSIC).

Mutual Information (MI):

$$I(X, Y) = D_{KL}(P_{XY} || P_X \otimes P_Y) = \iint p_{XY} \log \frac{p_{XY}}{p_X p_Y} dx dy$$

where the D_{KL} is the Kullback-Leibler divergence and p_{XY}, p_X, p_Y are the probability density functions.

Hilbert-Schmidt independence criterion (HSIC)

$$\text{HSIC}(X, Y) = \|m_k - m_{k_x} m_{k_y}\|_{\mathcal{H}}^2$$

For both MI and HSIC, they become 0 if and only if $X \perp\!\!\!\perp Y$. However, estimating them on a finite sample can be challenging in practice. Though we are unable to directly "check" the independence given only the observation, we may use the above dependence measures to characterize the independence of X, Y . In practice, they can be applied as test statistics in independent testing.

2.2.3 Conditional independence

In this subsection, we introduce the definition of conditional independence, which is a natural extension of independence with another variable Z . We also show some measures for conditional dependence.

Definition Given random variables X, Y, Z and we assume joint probability density function p_{XYZ} is continuous and p_X, p_Y, p_Z exist. X, Y are conditional inde-

pendent given Z , denoted as $X \perp\!\!\!\perp Y \mid Z$, is defined as

$$p_{XY|Z} = p_{X|Z}p_{Y|Z}$$

or equivalently

$$p_{XYZ} = p_{X|Z}p_{Y|Z}p_Z$$

$$p_{X|YZ} = p_{X|Z}$$

$$p_{Y|XZ} = p_{Y|Z}$$

$$p_{XYZ}p_Z = p_{XZ}p_{YZ}$$

Several characterizations of CI have been proposed and applied to CI tests [20]. We may start by considering a simple case when X, Y, Z are joint Gaussian distributed, the conditional independence of X, Y given Z is equivalent to the partial correlation being zero [14].

Partial Correlation

$$\rho_{XY,Z} = \frac{\text{cov}(X - \text{E}[X \mid Z], Y - \text{E}[Y \mid Z])}{\sqrt{\text{var}(X - \text{E}[X \mid Z]) \text{var}(Y - \text{E}[Y \mid Z])}}$$

where $\text{E}[\cdot|\cdot]$ and $\text{var}(\cdot)$ denote the conditional expectation and variance, respectively, and $\text{cov}(\cdot, \cdot)$ denotes the covariance between two variables. The property that $\rho_{XY,Z} = 0$ if and only if $X \perp\!\!\!\perp Y \mid Z$ only holds under the assumption that X, Y, Z follow a jointly Gaussian distribution. In general, partial correlation becoming zero does not necessarily imply conditional independence.

On the other hand, there are also some characterizations of CI unaffected by the assumption on the joint distribution of X, Y, Z . We will show the example of conditional mutual information, which is an extension of mutual information.

Conditional Mutual Information

$$I(X, Y | Z) = \iiint p_{XYZ} \log\left(\frac{p_Z p_{XYZ}}{p_{XZ} p_{YZ}}\right) dx dy dz$$

The following holds true for an arbitrary joint distribution of X, Y, Z .

$$I(X, Y | Z) = 0 \iff X \perp\!\!\!\perp Y | Z.$$

In Chapter 4, we discuss some other nonparametric CI tests based on different characterizations of CI, and propose a novel CI test.

3 | Convex clustering

3.1 Introduction

Clustering is a popular unsupervised learning task that involves exploring data to identify groups of similar objects. There are several traditional clustering methods, including hierarchical clustering, partitive clustering, and model-based clustering. More recently, researchers have been studying convex clustering [21, 22, 23], which is known for its global optimality due to the problem's convex formulation. Unlike methods like k -means, which require a predetermined number of clusters, convex clustering uses a tuning parameter to control the number of clusters in the output.

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

$$L(\mathbf{A}) = \frac{1}{2} \sum_{i=1}^n \|\mathbf{x}_i - \mathbf{a}_i\|_2^2 + \lambda \sum_{i < j} \omega_{ij} \|\mathbf{a}_i - \mathbf{a}_j\|_q. \quad (3.1)$$

In the above loss function, each $\mathbf{a}_i \in \mathbb{R}^p$ represents an alternative vector to represent the point $\mathbf{x}_i \in \mathbb{R}^p$. The \mathbf{A} is a matrix whose rows correspond to these alternative vectors. The L_q -norm, denoted by $\|\cdot\|_q$, is typically chosen as 1, 2, or ∞ [21]. The tuning parameter λ is a positive constant, and the weights ω_{ij} are chosen based on the input data. By solving the optimization problem, the optimal solution $\hat{\mathbf{A}} = (\hat{\mathbf{a}}_1, \dots, \hat{\mathbf{a}}_n)^T$ is obtained. To assign samples to the same cluster, we check whether the corresponding alternative vectors are equal. We assign the samples i and j to a same cluster if and only if $\hat{\mathbf{a}}_i = \hat{\mathbf{a}}_j$.

The solution path of the optimization problem has a meaningful visual interpretation known as the "clusterpath" (Figure 3.1). By varying the tuning parameter λ , the clusterpath shows how each point becomes merged along the path, provid-

ing rich information about the cluster structure of the data. Specifically, when $\lambda = 0$, each point occupies a unique cluster, and as λ increases, the clusters begin to coalesce. Eventually, for a sufficiently large λ , all the points coalesce into a single cluster. The clusterpath visualization is a valuable tool for exploring and understanding the clustering behavior of the proposed method.

In general, constructing the clusterpath involves a high computational cost. To solve (3.1), Hocking et al. [21] proposed to use three algorithms for different regularization terms corresponding to L_1 , L_2 , and L_∞ . Subsequently, general solvers for convex problems such as the alternating direction method of multipliers (ADMM) and the alternating minimization algorithm (AMA) [24] have been applied to solve (3.1) with L_1 and L_2 penalties. To obtain the clusterpath efficiently, Weylandt et al. [25] proposed the CARP algorithm, which uses a novel computational technique to approximate the path-wise visualizations with sufficient precision. For the L_1 penalty case with identical weights setting (i.e. $\omega_{ij} = 1$), Radchenko and Mukherjee [26] considered two efficient algorithms that successively merge the clusters in a bottom-up fashion or split the clusters in a top-down fashion to detect all the fusion or split events. Additionally, they studied the sample behavior of convex clustering with L_1 penalty and identical weights, providing theoretical support. However, their methods cannot estimate $\hat{\mathbf{A}}$ and therefore cannot provide a clusterpath.

In this chapter, we consider the same setting as Radchenko and Mukherjee [26]: L_1 convex clustering with identical weights. However, we propose a completely different approach and develop an efficient algorithm to handle the computational bottleneck in the convex clustering problem. Fortunately, for the problem (3.1) with the L_1 penalty, Hocking et al. [21] noted that the problem is separable along dimensions. In addition, for each dimension the sub-problem becomes the following general fused Lasso problem [2]. The problem (3.1) is decomposed into p

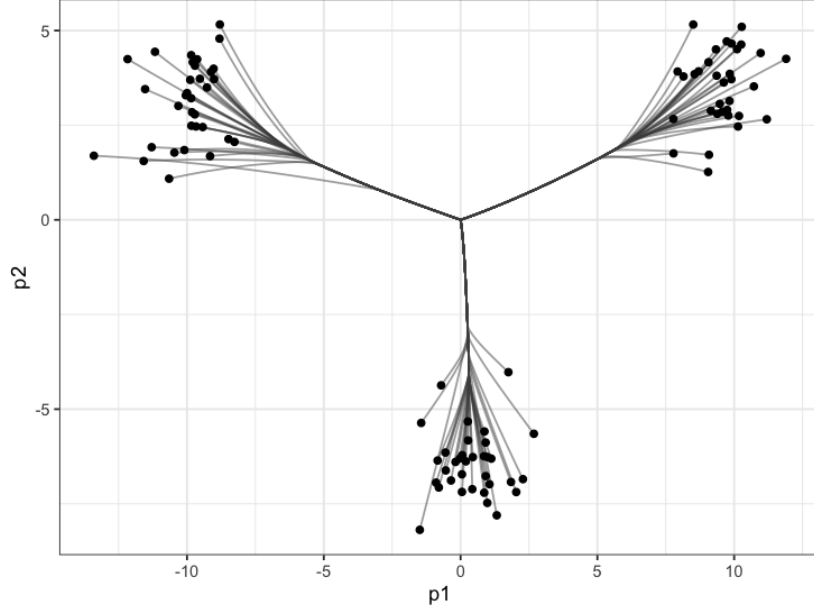


Figure 3.1: An example shows the clusterpath generated under L_1 norm. The clusterpath shows individual cluster centers start to merge and finish as a single cluster.

separate sub-problems as follows:

$$\min_{\mathbf{a} \in \mathbb{R}^n} \frac{1}{2} \sum_{i=1}^n (x_i - a_i)^2 + \lambda \sum_{i=1}^{n-1} \sum_{j=i+1}^n |a_i - a_j| \quad (3.2)$$

Hocking et al. [21] applied the FLSA algorithm [3] to solve the problem (3.2). However, based on our experience, it still remains very challenging for large-scale problems. Therefore, we employ a dynamic programming (DP) method to obtain the exact solution of the problem (3.2). Johnson [9] first proposed the dynamic programming method for the chain-graph fused lasso, or simply, the chain-structured 1d fused Lasso, which penalizes the neighbor terms:

$$\min_{\mathbf{a} \in \mathbb{R}^n} \frac{1}{2} \sum_{i=1}^n (x_i - a_i)^2 + \lambda \sum_{i=1}^{n-1} |a_i - a_{i+1}| \quad (3.3)$$

Although (3.2) and (3.3) may appear different at first glance, we demonstrate in this chapter that there is a close relation. Specifically, we begin by reformulating the sub-problem of convex clustering expressed in (3.2) as a weighted one-dimensional fused lasso problem in (3.3). This reformulation enables us to utilize a modified version of the dynamic programming method, which efficiently solves the problem in linear time. Our numerical experiments validate the effectiveness of our proposed method, which outperforms existing approaches. We summarize our contributions in detail:

- Based on an important observation regarding the case of identical weights, we demonstrate that each sub-problem of convex clustering can be transformed into a weighted 1d fused lasso problem.
- We utilize a DP algorithm to solve the reformulated problem. In addition, we refine the problem’s formulation to improve its efficiency further. Our proposed algorithm, C-PAINT, builds upon the DP approach to construct a full clusterpath. The 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 the later simulation, we show that the C-PAINT takes $\mathcal{O}(pnK)$, which is scalable to large datasets.

The remaining article is arranged as follows. Specifically, we begin by discussing related work in Section 3.2. In Section 3.3, we introduce the preliminaries and present some key properties of L_1 convex clustering, which will be utilized to reformulate problem (3.2) later on. Next, in Section 3.4, we outline our DP method and C-PAINT algorithm, which allow us to draw a full clusterpath. To evaluate the effectiveness of our approach, we report our experimental results in Section 3.5, which includes both synthetic and real data. Finally, in Section 3.6, we conclude the article. For more detailed information about the DP algorithm, please refer to

the supplementary material.

3.2 Related work

Convex clustering has several variants in the literature. For instance, Chi et al. [27] introduced convex bi-clustering, while Wang et al. [28] proposed sparse convex clustering, which enables simultaneous clustering and feature selection. In addition, Wang et al. [29] proposed robust convex clustering, which is designed to detect outlier features.

From the computational perspective, several theoretical studies have been done on convex clustering. For example, Tan and Witten [30] demonstrated that when identical weights are employed, convex clustering is closely related to single-linkage hierarchical clustering and the k -means method. Radchenko and Mukherjee [26] analyzed the asymptotic properties of the solution path and provided conditions for it to yield the true dendrogram under L_1 fusion penalty with identical weights. Furthermore, Zhu et al. [31] investigated the conditions required for convex clustering to recover clusters correctly.

From the computational perspective, studies are focused on fast computational methods for convex clustering. Lindsten et al. [22] proposed to use the off-the-shelf solver `CVX` to generate the solution path. Hocking et al. [21] introduced three algorithms for three different penalty norms (L_1 , L_2 , and L_∞). They used the FLSA algorithm for L_1 penalties. Chi and Lange [24] proposed to use 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. [32] proposed a semi-smooth Newton-based algorithm to solve the convex clustering problem. Weylandt et al. [25] proposed an ADMM-based approximation method to obtain a complete so-

lution path. Radchenko and Mukherjee [26] 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.

Despite its many competitive features, convex clustering remains computationally challenging due to its high computational burden.

3.3 Preliminaries

In this chapter, we use small bold letters to denote vectors, such as \mathbf{x} , and use ordinary letters to denote scalars, such as x .

At first, we discuss the decomposition of the L_1 convex clustering loss function into sub-problems by each dimension. Next, we introduce some essential properties about the loss function of each sub-problem, which will be useful for our problem reformulation in Section 3.4.

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 following convex clustering problem with L_1 fusion penalty and identical weights:

$$\begin{aligned} & \frac{1}{2} \sum_{i=1}^n \|\mathbf{x}_i - \mathbf{a}_i\|_2^2 + \lambda \sum_{i < j} \|\mathbf{a}_i - \mathbf{a}_j\|_1 \\ & = \sum_{k=1}^p \left[\frac{1}{2} \sum_{i=1}^n (x_{ik} - a_{ik})^2 + \lambda \sum_{i=1}^{n-1} \sum_{j=i+1}^n |a_{ik} - a_{jk}| \right] \end{aligned} \quad (3.4)$$

It is easy to notice that solving the minimization problem in (3.1) is equivalent to solving p separate sub-problems, one for each dimension. In the following discussion, we focus on the sub-problem (3.2).

Now, we introduce the following theorem and lemma that prepare us for the reformulation in the next section. [21] first shows that in the L_1 clusterpath, no split happens in the identical weights setting 3.3.1. In other words, for a small λ_1 , if two parameters become the same value $\hat{a}_i(\lambda_1) = \hat{a}_j(\lambda_1)$, they will become the same value i.e. $\hat{a}_i(\lambda_2) = \hat{a}_j(\lambda_2)$ for any bigger $\lambda_2 > \lambda_1$. Next, we have lemma 3.3.2 that the order of the original input is preserved in estimated centroids.

Theorem 3.3.1 (Hocking et al. [21]) *Taking $\omega_{ij} = 1$ for all i and j is sufficient to ensure that the ℓ_1 clusterpath contains no splits.*

Lemma 3.3.2 (Chiquet et al. [33]) *The absence of splits is equivalent to the preservation of the order along the path for problem (3.2).*

3.4 Proposed method

In this section, we demonstrate how to reformulate the problem (3.2) by substituting the penalty term in Section 3.4.1. Subsequently, we describe how to apply the DP method to solve the reformulated objective function, providing the specific modifications in Section 3.4.2. While Algorithm 1 presents a high-level overview of the DP algorithm, we provide additional details in Algorithm 2. Next, we propose the C-PAINT algorithm, which is based on the DP algorithm and is outlined in Algorithm 3 in Section 3.4.3. Finally, we analyze the time complexity of the proposed procedure in Section 3.4.4.

3.4.1 Idea

Our idea is simple: to reformulate each original sub-problem into an equivalent form. On the theoretical side, a direct consequence of Theorem 3.3.1 and Lemma 3.3.2 is that for problem (3.2), the order of the elements in \mathbf{x} is preserved in $\hat{\mathbf{a}}$. In

other words, the estimated centroids preserve the original order of the input data.

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

where the $x_{(i)}, i = 1, \dots, n$ are the 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 (3.2):

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^n |a_i - a_j|.$$

Suppose the order of (a_1, \dots, a_n) is known, then the absolute value can be removed. For example, let $n = 4$, then 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 each penalty term, we can rewrite it into the absolute values of the differences between neighbor items.

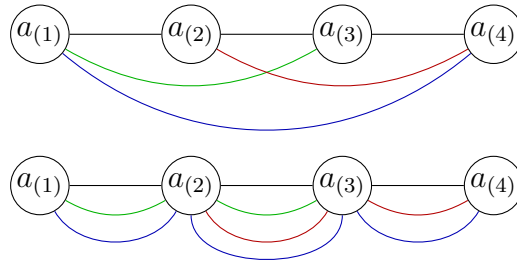


Figure 3.2: Replacement of the penalty graph. The edges are the absolute values of the differences between nodes. The total sums of the edges' lengths are identical for the two graphs, which inspires us to reformulate the penalty term.

Figure 3.2 shows a graphical representation of the transformation from a complete graph to a weighted chain graph. The complete graph is shown in the above figure, where each vertex represents a centroid parameter, and all pairs of vertices are connected by edges. The weighted chain graph is shown below, where each vertex

still represents a centroid parameter, but the vertices are connected in a linear chain. The edges between the vertices in the weighted chain graph represent the absolute differences between neighboring items, as discussed. And the number of edges becomes the weights. Inspired by this important observation, we have the following lemma 3.4.1.

Lemma 3.4.1 *Given the sequence (a_1, \dots, a_n) , we can sort it in a non-decreasing order $a_{(1)} \leq \dots \leq a_{(n)}$, then*

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^n |a_i - a_j| = \sum_{i=1}^{n-1} i(n-i) |a_{(i)} - a_{(i+1)}|. \quad (3.5)$$

Lemma 3.4.1 suggests the possibility to replace the penalty term in (3.2) with the right side one in (3.5). By the replacement, the original fused lasso problem is turned into a weighted chain-structured fused Lasso as follows:

$$\min_{\mathbf{a} \in \mathbb{R}^n} \frac{1}{2} \sum_{i=1}^n (x_i - a_i)^2 + \lambda \sum_{i=1}^{n-1} i(n-i) |a_{(i)} - a_{(i+1)}| \quad (3.6)$$

Next, we show how to apply the dynamic programming method [9] to solve the problem (3.6). 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. In the following subsection 3.4.2, we introduce the modified DP algorithm in further detail.

3.4.2 DP algorithm

Given a sequence of sorted data points $x_1 \leq x_2 \leq \dots \leq x_n$. Suppose the corresponding centroids are a_1, \dots, a_n . By Lemma 3.3.2, 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:

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

where $\lambda_i = \lambda i(n-i)$. Before showing the details about the dynamic programming algorithm, we 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 3.4.2 (Johnson [9]) *The function $h_k(b)$ is convex, differentiable, and piecewise quadratic.*

The b and \tilde{b} in the definition of ϕ_k correspond to the former centroid parameter a_k and the latter centroid a_{k+1} respectively. Once \hat{a}_{k+1} is known, by definition, \hat{a}_k can be expressed as a functional form of \hat{a}_{k+1} :

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

From the above theorem 3.4.2, 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 $\hat{a}_k = \phi_k(\hat{a}_{k+1})$

Algorithm 1: DP algorithm

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

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 $\hat{a}_k = \min(\hat{a}_{k+1}, U_k)$
 - 8 **end**
-

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

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

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 minimum between \hat{a}_{k+1} and U_k and assign it to \hat{a}_k . It is easy to see that once we find \hat{a}_n and U_1, \dots, U_{n-1} , we can obtain all the centroids by tracking back from $n-1, \dots, 1$. Algorithm 1 provides a high-level overview of the detailed 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 conditions, 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} . Specifically, when $b > U_{k-1}$, g_k is a line with the slope 1 and the intercept $\lambda_{k-1} - x_k$. On the other hand, when $b \leq U_{k-1}$, again, it becomes piecewise linear with a new knot point U_{k-2} , and so on. This iterative process continues until the last knot point U_1 . We notice that $g_{k-1}(b)$ includes the term $(b - x_{k-1})$, so the slope becomes steeper when the first condition $b \leq U_{k-1}$ is satisfied. Because g_k is a piecewise linear function, the key to finding 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 the 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, we need to be careful when searching for each U_k to ensure that it has a worst-case performance of $\mathcal{O}(n)$, which is achieved through the erase step in line 11 of Algorithm 2. In 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 may be challenging to understand. In order not to interrupt the flow of the chapter, we include an example of $n = 3$ in the supplementary materials, which we believe is helpful in understanding the algorithm.

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  $\beta$  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  $U^* \leftarrow (U^*, U_k), S^* \leftarrow (S^*, S_k), I^* \leftarrow (I^*, I_k)$ .
21 end
```

3.4.3 C-PAINT algorithm

The dynamic programming algorithm is intended for a single tuning parameter λ . However, it is of our interest to visualize the complete clusterpath. Using the proposition's result, we can first find the maximum value of λ_{\max} that yields a non-trivial solution. In other words, any tuning parameter larger than λ_{\max} results in just one cluster. Next, we propose the C-PAINT algorithm.

Proposition 3.4.3 (Radchenko and Mukherjee [26])

Given data $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{R}^n$. In the problem 3.2, 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 $\bar{\mathbf{x}} = \frac{1}{n} \sum_{k=1}^n x_k$.

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 because it simplifies the computational burden by reducing the number of tuning parameters that need to be considered. Instead of considering all possible values of λ , we only need to consider a sequence of λ up to λ_{\max} . This reduces the computational complexity and makes the algorithm more efficient. In Algorithm 3, an arithmetic sequence is used to select the values of λ . However, it is also possible to use a geometric sequence. The choice of the sequence depends on the specific problem.

3.4.4 Time complexity

To analyze the time complexity of the C-PAINT algorithm, we need to consider the number of operations involved in each step.

In Algorithm 2, U_1 can be founded in $\mathcal{O}(1)$. In finding U_2, \dots, U_{n-1} , lines 2-3, lines 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 need to enter

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
```

the inner 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, Algorithm 2 is $\mathcal{O}(n)$.

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

3.5 Experimental evaluation

In this section, we will first compare the runtimes of different methods for L_1 convex clustering. The C-PAINT is compared with several representative methods: FLSA [3], ADMM, AMA [24], CARP [25], and the FUSION algorithm [26]. The results demonstrate the effectiveness and efficiency of our proposed method in finding clusterpaths. As the proposed method is a novel optimization method that yields the exact solution, we focus on showing the visualizations of the obtained clusterpaths, which clearly illustrate how the clusters evolve along the path. In the synthetic data example, we generated five clusters with varying shapes, and the obtained clusterpath shows the merging of the clusters along the path. In the real data examples, we applied C-PAINT and other methods to relatively small datasets and presented the runtimes. We also demonstrated the applicability of C-PAINT to larger datasets, which were not possible for existing methods.

3.5.1 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 3.3.1. The ADMM and AMA are implemented in the `cvxcluster` 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 `fusioncluster` R package. To make a fair comparison, we implemented the code in Rcpp by ourselves and made some modifications to accelerate the algorithm. Because Rad-

chenko and Mukherjee [26] 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 runtimes.

3.5.2 Time comparison

The simulated data in this experiment consists of points sampled from a Gaussian mixture model with three components in \mathbb{R}^2 . We generate datasets with sizes ranging from 100 to 50,000 points. For each dataset, we construct a λ sequence consisting of $K = 10$ values, evenly spaced between 0 and λ_{\max} . The reported runtimes are the means of 30 replications. Due to computational limitations, we only allowed the CARP algorithm to run on data sets with up to 500 points and restricted the FLSA, ADMM, and AMA algorithms to data sets with up to 1000 points. In contrast, we ran the FUSION algorithm and the proposed method on larger data sets of 5000, 10000, and 50000 points. While it is possible to solve each sub-problem in parallel for the L_1 case in C-PAINT, FLSA, and FUSION, we did not pursue it in our experiments.

Figure 3.3 displays the time comparison results. The x-axis corresponds to the sample size n , while the y-axis represents the runtimes in seconds on a logarithmic scale.

We notice for the identical weights and L_1 penalty setting, CARP is slower than

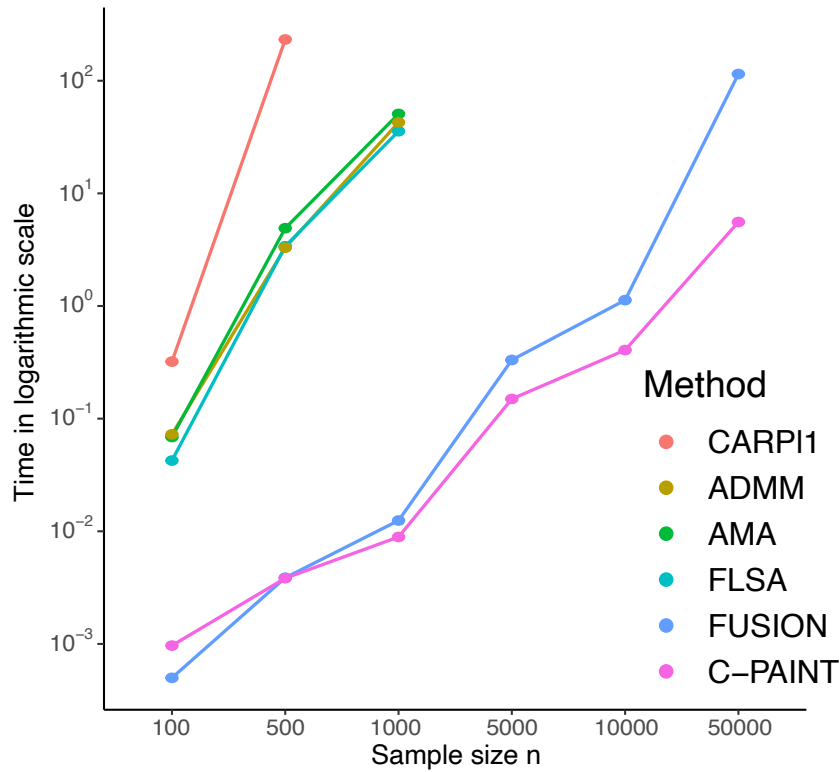


Figure 3.3: Comparison of the runtimes in computing clustering solution path in logarithmic scale. Each line represents the runtimes of different methods.

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 highly efficient but is unable to create a clusterpath visualization because it does not estimate the exact value of centroids with a given tuning parameter λ . In terms of visualizing the clusterpath and performing clustering, C-PAINT is the most efficient method. In particular, C-PAINT can find the full solution path of 10^7 samples in \mathbb{R}^2 within two minutes. Additionally, the simulation results demonstrate that the runtimes of C-PAINT increase linearly, confirming the time complexity analysis presented in Section 3.4. When the sample size n becomes large, C-PAINT is generally faster than FUSION. The possible explanation for this is that FUSION has to go through every fusion event, which results

in decreased efficiency as the sample size increases. On the other hand, the proposed method independently performs the DP algorithm with each single tuning parameter λ . To summarize, in terms of finding the clusterpath, C-PAINT outperforms CARP, FLSA, ADMM, and AMA, and can handle large-scale problems effectively. Please refer to the Appendix for detailed results.

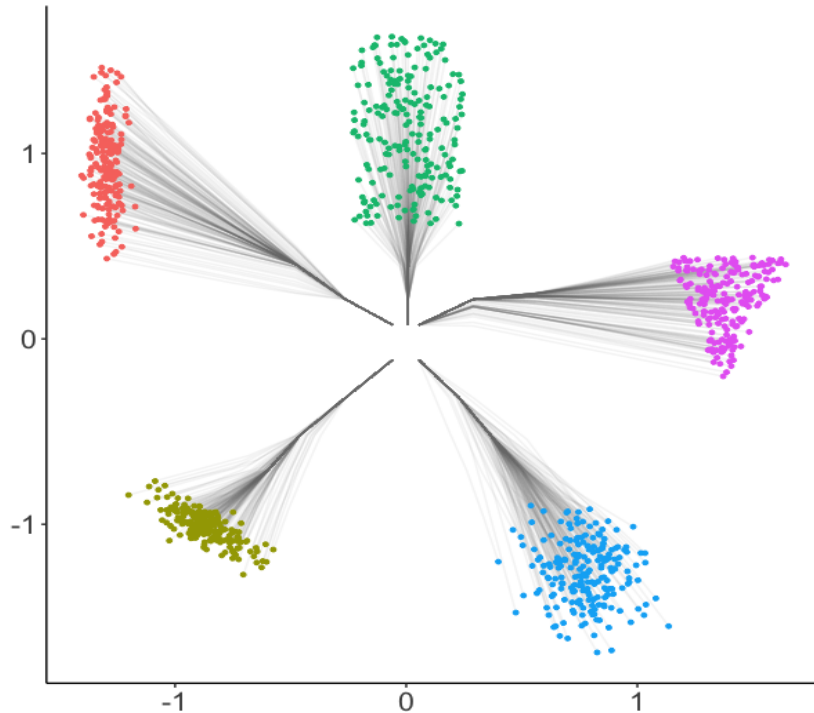


Figure 3.4: Visualization of the clusterpath generated with a λ sequence with length $K = 10$. The colors show the clustering results with the biggest tuning parameter of the lambda sequence. The threshold γ are set to be 10^{-6} .

3.5.3 Synthetic dataset

We generate the synthetic data into five clusters with different shapes, each including 200 data points. For 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 the same cluster.

In Figure 3.4, both the clustering results and clusterpath are presented in the same plot. The colors represent the 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, convex clustering successfully separates different clusters.

3.5.4 Small real dataset

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

- **Lymphoma** [34] dataset includes 62 samples categorized into three different lymphoma types.
- **Gene expression** [35] dataset includes the gene expression features of 801 samples with four different types of tumor as labels.

To enhance the visualization of high-dimensional datasets, we first utilize the UMAP [36] to reduce the dimensions to two. The dimensionality reduction is carried out using the `umap` function in the `uwot` R package. Subsequently, we apply C-PAINT to the projected coordinates and use the original labels to denote the clusters in different colors.

Both the full clusterpaths and runtimes are reported in the table 3.1. Here, we only report the ADMM since the ADMM, AMA, FLSA have similar runtimes, as shown in Figure 3.3. From the result, we can see C-PAINT is much faster than other methods.

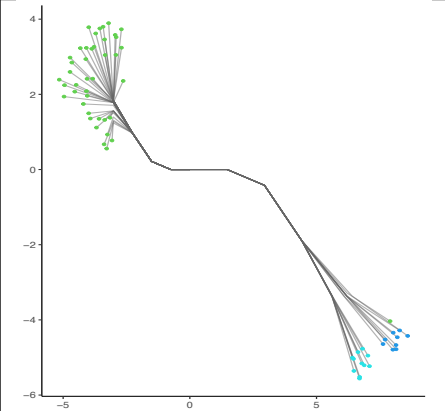
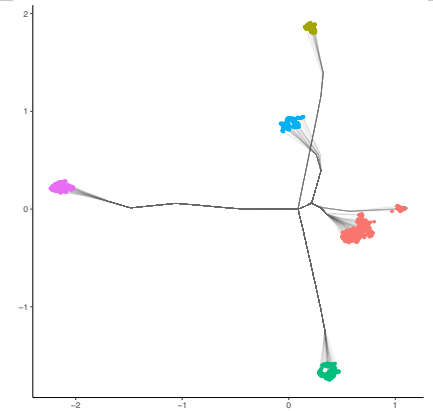
Lymphoma	Gene expression
	
Sample size : $n = 62$	$n = 801$
runtime (second)	
CARP: 6.6×10^{-2}	223
ADMM: 1.1×10^{-2}	8.6
C-PAINT: 4.7×10^{-4}	0.012

Table 3.1: Visualization of the clusterpath generated with a λ sequence with length $K = 10$. The colors show the original labels of the samples. runtimes are the means over 30 replications.

3.5.5 Large real dataset

Next, 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 `sndata` R package.
- **RNA sequence** [37] multi-datasets include 5683 cells consisting of 11 cell types and differentially expressed genes as their features.
- **Anuran (frog) calls** [38] 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.

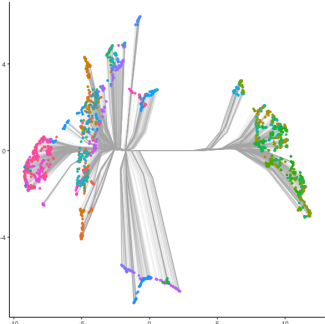
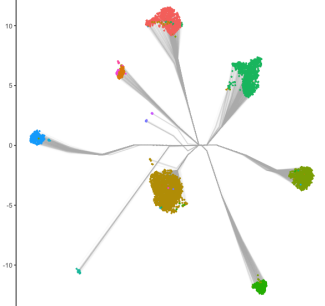
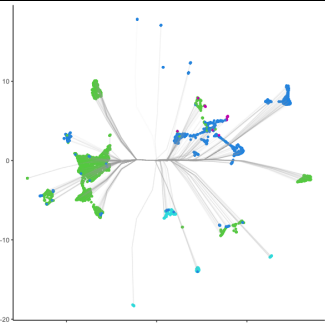
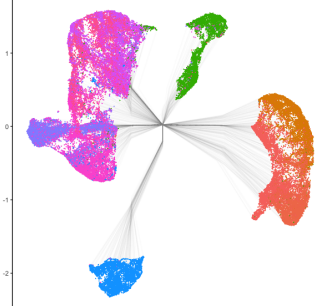
Frey faces	RNA sequence
	
Sample size : $n = 7195$	$n = 70000$
runtime (s) : 8.4×10^{-3}	2.6×10^{-2}
Anuran (frog) calls	Fashion-MNIST
	
Sample size : $n = 1965$	$n = 5683$
runtime (s) : 5.6×10^{-2}	15.8

Table 3.2: 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 runtimes of each real dataset are the means over 10 replications.

-
- **Fashion-MNIST** [39] dataset consists of 70000 grayscale images of items such as T-shirt, Trouser and Bag. Fashion-MNIST includes 10 labels and each label has the same number of items.
 - **Kuzushiji-MNIST** [40] dataset consists of 70000 grayscale images of hiragana characters in Japanese. Fashion-MNIST includes 10 labels and each label has the same number of characters.

Here we focus on showing the clusterpaths on large datasets. Table 3.2 shows the results of the first four datasets. Both the sample sizes and the runtimes of the C-PAINT are presented. We choose the Kuzushiji-MNIST dataset [40] to explain in further detail.

Kuzushiji-MNIST is a drop-in replacement for the MNIST dataset, consisting of ten rows of Japanese Hiraganas. 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 Figure 3.5 that except for the Hiragana Ha, others have two or more variants that look quite unlike. To be more specific, on the right side of the clusterpath, we can see that Ha and Tsu merge early on as they share a high degree of similarity in their image projections.

3.6 Summary

We have developed a new algorithm for L_1 convex clustering. To the best of our knowledge, it is the first time that dynamic programming has been applied to the convex clustering problem. By formulating the sub-problems for each dimension as weighted one-dimensional fused lasso problems, we can apply a dynamic programming algorithm to solve them efficiently.

In order to visualize the clusterpath, we proposed the C-PAINT based on the dy-

dynamic programming algorithm. The time complexity of C-PAINT is $\mathcal{O}(pn \log n) + \mathcal{O}(pnK)$. However, in practical applications, the run time of our algorithm grows linearly as the sample size n and dimensionality p increase. The proposed algorithm is highly efficient and outperforms existing algorithms in terms of scalability. For the important special case of L_1 convex clustering with identical weights, the simulation results show our proposed method overcomes the computational bottleneck, 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>.

A possible direction for future work is to explore a more general structure of weights. Currently, C-PAINT algorithm is an efficient method, but it is limited to the identical weight setting. On the other hand, it is pointed out that general weights result in advanced performance [41] e.g. weights that have a k-Nearest Neighbor graph structure and assigned the values using Gaussian kernel.

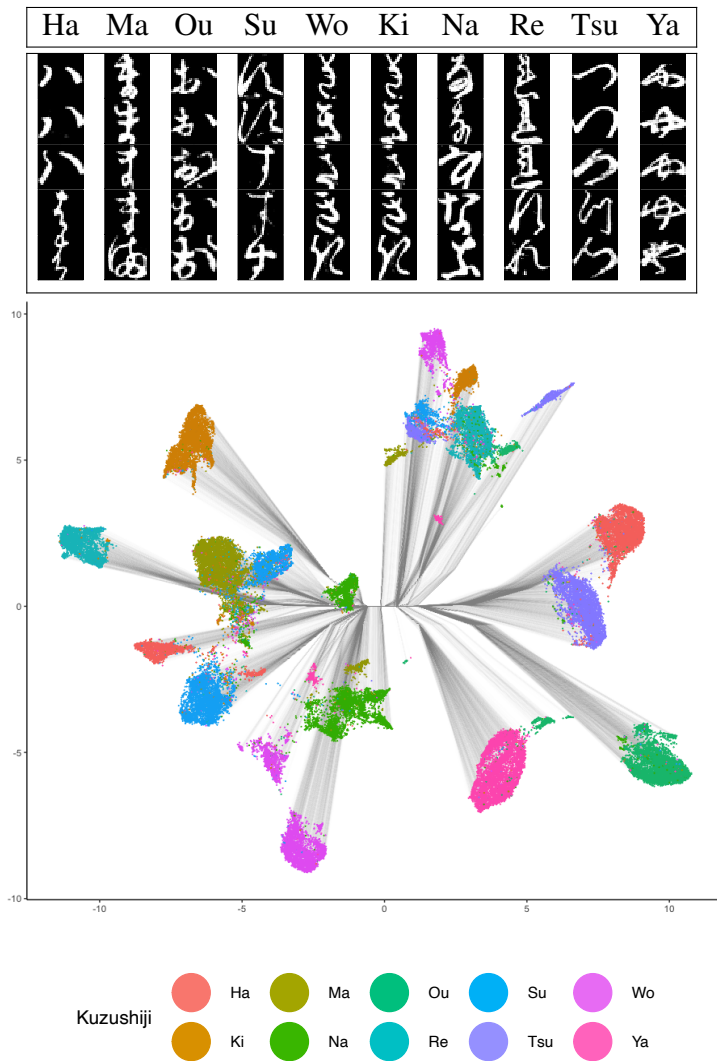


Figure 3.5: Some representative variants of kuzushiji and the legend are shown on the top. The middle figure show the visualization of the clusterpath of kuzushiji-MNIST. The length of the λ sequence is set to be $K = 5$. The mean runtime is 11.578 seconds over ten replications.

4 | Conditional independence test

4.1 Introduction

Conditional Independence (CI) test is a statistical hypothesis test used to determine if variables X and Y are conditionally independent, given another variable Z fixed. If variable X and Y are conditional independent given Z , we denote it as $X \perp\!\!\!\perp Y \mid Z$. The CI test is an important tool to analyze the relationship between variables and is applied in causal discovery [42, 43, 13].

The CI test is easy to perform with a large sample size n and discrete variable Z . At that case, we can test the independence of X and Y for each possible value of Z . On the other hand, if X , Y , and Z have a joint Gaussian distribution, then CI of X, Y given Z is equivalent to the zero partial correlation between X and Y given Z [14], which can be easily tested as well. In this chapter, we aim to examine the conditional independence of X , Y , and Z without making any assumptions about their joint distribution, regardless of whether they are continuous or discrete variables. However, the challenge arises when the dimension d_Z grows, which results in the curse of dimensionality [44]. When Z is a set of d_Z variables or any d_Z -dimensional random vector, the problem becomes more complex.

One of the major challenges in CI tests is the requirement to obtain a sample from the null distribution $H_0 : X \perp\!\!\!\perp Y \mid Z$. In statistical hypothesis testing, it is generally necessary to determine the distribution of the test statistic under the null hypothesis H_0 . However, in the case of CI tests where we only have access to the observations, it is impossible to know the exact distribution of any test statistic under the null hypothesis $H_0 : X \perp\!\!\!\perp Y \mid Z$. To address the problem, there are two popular approaches for obtaining an approximate null distribution:

- Asymptotic method

One approach utilizes the asymptotic distributions of the test statistics [45, 18, 46]. For some test statistics, their asymptotic distributions are derived. And for such cases, the asymptotic distribution can be used to approximate the null distribution. Though these asymptotic distributions can be generated efficiently, they may become less accurate when the sample size n is small or with a high-dimensional Z [47, 48].

- Permutation method

The other approach is by permuting the observed samples. Suppose samples $\{x_i\}, \{y_i\}, \{z_i\}, i = 1, \dots, n$ are available for X, Y, Z . In independence test where $H_0 : X \perp\!\!\!\perp Y$, though X and Y in each pair $(x_1, y_1), \dots, (x_n, y_n)$ are not independent, we may regard X and Y of shifted pairs of $(x_1, y_2), \dots, (x_{n-1}, y_n), (x_n, y_1)$ to be independent. Thus we can compute the test statistic values on the shifted pairs, which mimic H_0 and obtain a histogram as an approximated null distribution. However, in CI test, as the conditioning set Z exists, we cannot shift $\{x_i\}, \{y_i\}, \{z_i\}$ in order to make them conditional independent [49, 47].

In this chapter, we propose a new CI test including a novel test statistic and combined with a local bootstrap method to sample from the $H_0 : X \perp\!\!\!\perp Y \mid Z$. In CI tests, many test statistics are calculated based on a direct evaluation of the conditioning set Z . This can be challenging, particularly when Z is high-dimensional or has a complex density. Our proposed test statistic does not directly rely on the conditioning set Z , which mitigates the issue of the curse of dimensionality. The test is expected to be more robust in situations where the conditioning set is high-dimensional. The experimental results show that our proposed test has comparable performance when the conditioning set Z is low-dimensional and notably outperforms other methods when Z is high-dimensional. Moreover, our proposed

method can be computed efficiently as the sample size n and the dimension of Z grow. We summarize our main contributions in detail:

- We proposed a novel test statistic that is calculated in the following way: first, we divide the variable Z into multiple local clusters. Next, we measure the unconditional independence within each cluster, and finally, we aggregate these unconditional independence measures to obtain a single statistic to measure conditional independence. In particular, we apply the k -means algorithm to perform clustering on Z , and for each cluster, we use the Hilbert-Schmidt Independence Criterion (HSIC) [17] as the measure of unconditional independence. This approach allows us to avoid direct access to Z and thus alleviate the curse of dimensionality, making the proposed method more robust for high-dimensional conditioning sets.
- We apply a local bootstrap method to mimic sample from $H_0 : X \perp\!\!\!\perp Y \mid Z$. We extended the local bootstrap strategy in [50]. When combined with the proposed test statistic, the local bootstrap method shows good performance and provides higher power on both linear and non-linear cases. The local bootstrap method can be applied not only to the proposed test statistic but also to other CI tests.

This chapter is organized as follows. In Section 4.2, we discuss some related work on the CI testing. In Section 4.3, we introduce the notations and provide an overview of the HSIC, a kernel-based measure of unconditional independence. In Section 4.4, we present the test procedure and explain the test statistic and the local bootstrap method in detail. In Section 4.5, we compare our proposed method with other representative CI tests using synthetic data. Finally, we summarize our results in Section 4.6.

4.2 Related work

Recently, numerous nonparametric methods have been proposed for CI testing. Many test statistics are constructed by embedding distributions in reproducing kernel Hilbert spaces (RKHS). Fukumizu et al. [49] proposed a measure of CI based on cross-covariance operators. However, its asymptotic distribution under the null hypothesis is unknown, and the bin-based permutation degrades as the dimension of conditioning variable Z grows. Later, KCIT was proposed Zhang et al. [18] based on the partial association of functions in RKHS. KCIT has the advantage that its asymptotic distribution is known, which can be efficiently approximated. To improve KCIT for testing on large-scale datasets, Strobl et al. [46] proposed RCIT and RCoT to use random Fourier features to approximate KCIT efficiently. Huang et al. [51] proposed a Kernel Partial Correlation (KPC), a generalization of partial correlation to measure conditional dependence. Beyond kernel-based methods, Runge [48] used a Conditional Mutual Information (CMI) estimator as the test statistic and proposed a k-nearest neighbor-based permutation to generate samples from the null distribution. Shah and Peters [52] proposed a generalized covariance measure (GCM) as the test statistic based on the regression method. On the other hand, CI can be turned into other problems. Doran et al. [47] turned the CI test into a two-sample test by finding a permutation matrix and measuring the Maximum Mean Discrepancy (MMD [19]) between the two distributions. Sen et al. [53] proposed a method called CCIT which turned the CI test into a classification problem. In [47, 53], they both have an additional sampling step involving data-splitting, which potentially reduces the power when the dataset is small. Some other model-powered methods also make use of deep learning: GAN [54, 55] and Double GAN [56].

While a nonparametric CI test makes no assumption about the joint distribution

of X, Y, Z , imposing additional assumptions can help simplify the problem. One of the mild assumptions is that X and Y are functions of the variable Z , plus an additive independent noise term with a zero mean:

$$X = f(Z) + \varepsilon_x, \quad Y = g(Z) + \varepsilon_y,$$

If the estimated noise terms are independent $\varepsilon_x \perp \varepsilon_y$, we conclude that $X \perp Y \mid Z$ [57, 58, 59, 60, 61]. The methods in this category find a regression function and then test for the unconditional independence of the residuals.

For some current different characterizations of CI, see for example, [20]. From a theoretical perspective, Shah and Peters [52] proved there exists no universally valid CI testing for all CI cases. In other words, no CI test can control Type-I error for all the CI cases while having a higher power against any alternative. However, a desirable CI test is supposed to be computationally efficient and widely applicable for different linear and non-linear cases.

4.3 Background on kernel methods

For random variables X, Y, Z , we use $x \in \mathcal{X}, y \in \mathcal{Y}, z \in \mathcal{Z}$ to denote their observed samples, and use $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ to denote the associated domains. We consider a positive-definite kernel $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$. The k has a corresponding a Hilbert space \mathcal{H} and a feature map $\Psi : \mathcal{X} \rightarrow \mathcal{H}$ such that

$$k(x_1, x_2) = \langle \Psi(x_1), \Psi(x_2) \rangle_{\mathcal{H}}$$

for $x_1, x_2 \in \mathcal{X}$. $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ is the inner product of a reproducing kernel Hilbert space \mathcal{H} .

Let $k_{\mathcal{X}}$ and $k_{\mathcal{Y}}$ be kernels on \mathcal{X} and \mathcal{Y} , and $\mathcal{H}_{k_{\mathcal{X}}}$ and $\mathcal{H}_{k_{\mathcal{Y}}}$ be the corresponding RKHSs. Gretton et al. [17] defined the Hilbert-Schmidt independence criterion (HSIC) which can be viewed as the squared MMD between a measure P_{XY} of X, Y and the product $P_X P_Y$ of measures P_X, P_Y . HSIC has been well studied as a test statistic in independence test [17, 18, 19]. For a characteristic kernel, the $\text{HSIC}(X, Y)$ is zero if and only if $P_{XY} = P_X P_Y$, which indicates $X \perp\!\!\!\perp Y$.

The HSIC is defined as follows.

$$\begin{aligned} \text{HSIC}(X, Y) &:= \|m_k - m_{k_{\mathcal{X}}} m_{k_{\mathcal{Y}}}\|_{\mathcal{H}}^2 \\ &= \|\mathbb{E}_{XY}[k_{\mathcal{X}}(X, \cdot)k_{\mathcal{Y}}(Y, \cdot)] - \mathbb{E}_X[k_{\mathcal{X}}(X, \cdot)]\mathbb{E}_Y[k_{\mathcal{Y}}(Y, \cdot)]\|_{\mathcal{H}}^2, \end{aligned}$$

where the \mathcal{H} is the corresponding RKHS of the kernel $k := k_{\mathcal{X}}k_{\mathcal{Y}}$ defined by

$$k((x, y), (x', y')) = k_{\mathcal{X}}(x, x')k_{\mathcal{Y}}(y, y')$$

for $(x, y), (x', y') \in \mathcal{X} \times \mathcal{Y}$.

$\text{HSIC}(X, Y)$ is known to have an alternative expression:

$$\text{HSIC}(X, Y) = \mathbb{E}_{XYX'Y'}[\mathbf{C}(X, Y, X', Y')] \quad (4.1)$$

where $\mathbf{C}(X, Y, X', Y')$ is

$$\left[k_{\mathcal{X}}(X, X') - \mathbb{E}_{X''}[k_{\mathcal{X}}(X, X'')] \right] \left[k_{\mathcal{Y}}(Y, Y') - \mathbb{E}_{Y''}[k_{\mathcal{Y}}(Y, Y'')] \right], \quad (4.2)$$

and (X', Y') are independent copies of (X, Y) .

Given data points $(x_1, y_1), \dots, (x_n, y_n)$, we consider the following estimator [17]:

$$\widehat{\text{HSIC}}(X, Y) = \frac{1}{n^2} \text{tr}(\mathbf{K}_X \mathbf{H} \mathbf{K}_Y \mathbf{H}) \quad (4.3)$$

where $(\mathbf{K}_X)_{ij} = k(x_i, x_j)$, $(\mathbf{K}_Y)_{ij} = k(y_i, y_j)$, $\mathbf{H} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^T$, and $\mathbf{1}$ is a n vector of ones. For HSIC, the asymptotic distribution under the null hypothesis $H_0 : X \perp\!\!\!\perp Y$ is derived [17, 18]. When we use it in the unconditional independence test, we can approximate the null distribution by using either permutation or its asymptotic distribution. Intuitively, we expect an estimator of HSIC to be a small value when $X \perp\!\!\!\perp Y$.

4.4 Proposed method

This section shows our proposed procedure for performing the CI test. First, we present a novel test statistic. The proposed test statistic is a kernel-based measure using characteristic kernels as a default choice, i.e., Gaussian kernel. Next, we explain the local bootstrap algorithm to generate samples from $H_0 : X \perp\!\!\!\perp Y \mid Z$. The local bootstrap algorithm sample x, y independently from discrete distributions, which mimic the distribution of H_0 . We repeatedly find the test statistics on the samples to obtain a histogram to calculate the p-value. We summarize the proposed test in Algorithm 4. Finally, we discuss the effect of parameters and provide a time complexity analysis of the overall procedure.

By definition, the conditional independence of X and Y given Z means variables X, Y are independent for any fixed value of Z , denoted as $X \perp\!\!\!\perp Y \mid Z$. Here, we use the notation $\text{HSIC}(X, Y \mid Z = z) := \mathbb{E}_{XYX'Y'}[C(X, Y, X', Y') \mid Z = z]$ to represent the HSIC on (X, Y) with fixed Z value, where the (X', Y') are copies of (X, Y) .

$$\begin{aligned} X \perp\!\!\!\perp Y \mid Z &\iff X \perp\!\!\!\perp Y \mid Z = z, \forall z \in \mathcal{Z}. \\ &\iff \text{HSIC}(X, Y \mid Z = z) = 0, \forall z \in \mathcal{Z}. \end{aligned}$$

As a direct result, we have the following proposition.

Proposition 4.4.1 (Characterization of CI)

$$X \perp\!\!\!\perp Y \mid Z \iff \int \text{HSIC}(X, Y \mid Z) d\mu(Z) = 0 \quad (4.4)$$

where $\mu(Z)$ is a probability measure on Z .

Proof sketch: By the definition of HSIC, $\text{HSIC}(X, Y \mid Z = z) = 0$ always takes non-negative values. Thus, for a characteristic kernel, the integral becomes zero if and only if $\text{HSIC}(X, Y \mid Z = z) = 0, \forall z \in \mathcal{Z}$, which indicates $X \perp\!\!\!\perp Y \mid Z$.

Thus, we consider measuring conditional dependence using the marginal unconditional dependence measures. However, it is unrealistic to assume that for all observations z , we have enough (x, y) pairs that share the same z values. As an alternative, we combine the clustering technique to divide Z into subgroups. Thus, observations of Z are grouped into different clusters with similar z values. As a result, we consider the following procedure to find our test statistic:

1. Perform clustering algorithm to subdivide Z into M clusters.
2. Measure the unconditional dependence of the (X, Y) pairs in the m -th cluster using estimators $\widehat{\text{HSIC}}_m(X, Y)$.
3. Find the sum of values as a single number

$$T = \sum_{m=1}^M \widehat{\text{HSIC}}_m(X, Y). \quad (4.5)$$

We use the sum of the local unconditional dependence measure as the conditional dependence measure, which is similar in spirit to [62]. Margaritis [62] considers dividing a univariate $Z \in \mathbb{R}^1$ into local bins and using the product of the local measures. Our method applies to a high-dimensional Z and takes the sum of

kernel-based measures.

Given data $\{(x_i, y_i, z_i)\}, i = 1, \dots, n$, we divide them into M clusters based on the value of Z using k -means. Let the index set of m -th cluster be C_m , and $C_m \subset \{1, \dots, n\}, \cup_{m=1}^M C_m = \{1, \dots, n\}, C_i \cap C_j = \emptyset, \forall i, j \in \{1, \dots, M\}$. The estimator $\widehat{\text{HSIC}}_m$ in (4.5) is

$$\widehat{\text{HSIC}}_m(X, Y) = \frac{1}{|C_m|^2} \text{tr}(\mathbf{K}_X^{(m)} \mathbf{H} \mathbf{K}_Y^{(m)} \mathbf{H})$$

where $|C_m|$ is the size of C_m , and $\mathbf{K}_X^{(m)}$ and $\mathbf{K}_Y^{(m)}$ are the corresponding kernel matrices for samples $(x_i, y_i), i \in C_m$. The conditioning set Z is only used in deciding the local clusters in the first step of the procedure. By doing that, we avoid a direct evaluation of the high-dimensional conditioning set Z .

4.4.1 Local bootstrap

In this subsection, we show a local bootstrap method to sample from $H_0 : X \perp\!\!\!\perp Y \mid Z$. We calculate the test statistic on generated samples multiple times and obtain a histogram to approximate the distribution of the test statistic under H_0 , which completes the CI test. The key in sampling is to break the dependence between X and Y while keeping the dependence between (X, Z) and (Y, Z) . An example of an ideal CI permutation is explained in Figure 4.1.

In the above Figure 4.1, we first divide different bins (green, red, and blue), and each bin \tilde{x} includes samples that have the same z . From that we fix y and z and shuffle x within each bin with some permutations (π_1, π_2, π_3) to generate new data (\tilde{x}, y, z) . An ideal permutation successfully generates samples that keep the dependence between (X, Z) and (Y, Z) while satisfying $X \perp\!\!\!\perp Y$. However, it is impossible to perform the ideal permutation in practice because we do not have

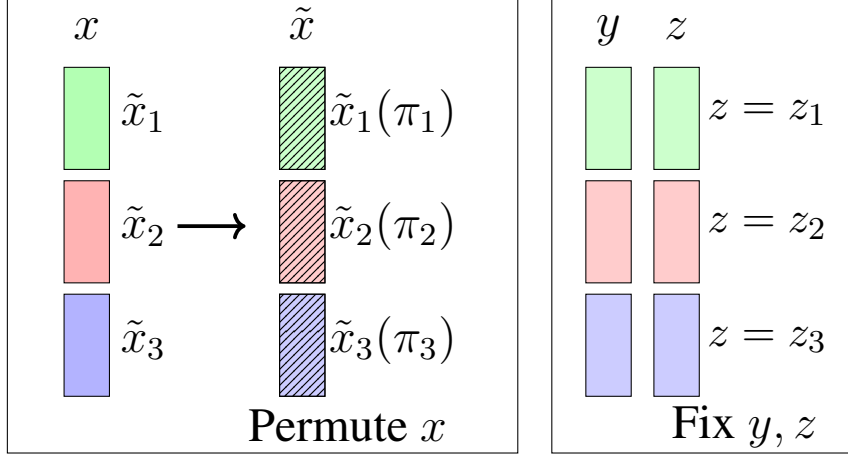


Figure 4.1: An ideal permutation in CI.

enough samples that have the same z values.

As an alternative approach, we use a local bootstrap. First, given with different z^* , we generate (x^*, y^*) independently from the following discrete distribution:

$$\begin{aligned}
 x^* &\sim \hat{G}_{x|z^*} : \begin{pmatrix} x_1 & x_2 & \dots & x_n \\ w_1 & w_2 & \dots & w_n \end{pmatrix}, \\
 y^* &\sim \hat{G}_{y|z^*} : \begin{pmatrix} y_1 & y_2 & \dots & y_n \\ w_1 & w_2 & \dots & w_n \end{pmatrix},
 \end{aligned} \tag{4.6}$$

where $w_j = \frac{K(z_j - z^*/\gamma)}{\sum_{j=1}^n K(z_j - z^*/\gamma)}$ is the probability to sample the index j .

This local bootstrap strategy is an extension of [50], originally designed to sample (x, y) according to a regression model. We extend it using a Nadaraya-Waston kernel estimator [63] to assign the weights for indexes to be sampled. If z_j is close to z^* , the w_j is large, and the index j is more likely to be sampled. Moreover, the x^* and y^* are sampled independently. Thus, it is less likely for x_j and y_j to be sampled simultaneously, which breaks the dependence between X and Y . Shi [50] suggested that the bandwidth γ should be varied for different z^* . Here, we

Algorithm 4: Local Bootstrap

Input: Data $(x_i, y_i, z_i), i = 1, \dots, n$.**Output:** New samples: $(x_i^*, y_i^*, z_i^*), i = 1, \dots, n$

- 1 **for** $i \leftarrow 1$ **to** n **do**
 - 2 Let $z_i^* = z_i$
 - 3 Sample $x_i^* \sim \hat{G}_{x|z_i^*}, y_i^* \sim \hat{G}_{y|z_i^*}$.
 - 4 **end**
-

Algorithm 5: Test

Input: Data $(x_i, y_i, z_i), i = 1, \dots, n$.Cluster number M .Times to repeat K .**Output:** p -value.

- 1 Find M Clusters.
- 2 Estimate the T .
- 3 **for** $k \leftarrow 1$ **to** K **do**
- 4 Generate samples with Algorithm 4
- 5 Estimate T_i on the generated samples.
- 6 **end**
- 7 Compute the p value:

$$p = \frac{1}{K} \sum_{k=1}^K \{T_k \geq T\}.$$

narrow the candidates from $1, \dots, n$ to 10-nearest neighbors of each z^* , and let the local bandwidth γ of z^* be the squared Euclidean distance between z^* and its 10-th nearest neighbor.

The local bootstrap is summarized in Algorithm 4. We generate samples multiple times and calculate the test statistic values T . We repeat it for K times on the generated samples and calculate the p -value based on the obtained histogram. We reject H_0 if the p value is smaller than a predefined significance level. Otherwise, we accept H_0 . We summarize the procedure in Algorithm 5.

4.4.2 Effect of M

The choice of the cluster number M can have an impact on the performance of the test. It is essential to choose M properly to ensure that the pairs (x, y) in each local cluster have similar z values while also ensuring enough pairs in each local cluster to obtain an accurate estimate of the local HSIC. The choice of the number of clusters M also impacts the computational complexity. A smaller M would result in larger clusters on average, requiring more time to calculate the local HSICs. In practice, we use k -means to decide the clusters and fix the average cluster size. We let M be $\lceil n/\bar{C} \rceil$, where $\lceil x \rceil$ takes the least integer that is not smaller than x and \bar{C} is the predefined average cluster size.

4.4.3 Complexity analysis

We provide an analysis of the time complexity of the test procedure. Initially, our method finds M clusters of Z using k -means. At the same time, the local bootstrap method requires the weights to be calculated once for each observation of Z . Both of these steps are calculated once and take little time. The major computational cost is in finding the histogram, where the test statistic T is calculated repeatedly for K times. Estimating a single T takes $\mathcal{O}(M|\tilde{C}|^2)$ operations, where M is the number of clusters, and $|\tilde{C}|$ is the maximum set size among all clusters and is smaller than n . This is repeated K times over generated samples to obtain the histogram of the null distribution. Overall, the proposed test takes $\mathcal{O}(Mn^2K)$. The bootstrap part can be easily parallelized to promote the speed further, but it is beyond the scope of the paper.

4.5 Experiments

In this section, we compare the proposed methods with other nonparametric CI tests. Our proposed method is denoted as **BHSIC** (Bundle of HSICs). We evaluate the Type-I error rate, Type-II error rate, and runtime of the methods. A good CI test should have a lower Type-II error rate and be computationally efficient. We compare with some representative methods: **KCIT** [18], **RCIT**, **RCoT** [46], **CCIT** [53], and **CMiknn** [48]. Details about these methods can be found in Section 4.2. All methods have source codes available online. We compare how these methods scale with sample size and the dimension of Z instead of a direct comparison of runtimes.

We aim to evaluate the performance of the methods under different scenarios. For our simulations, we consider two models: a simple linear regression model and a post-nonlinear noise model. The post-nonlinear noise model is a commonly used setting in evaluating CI tests [18, 46, 48], and the functional forms of X and Y on Z are as follows:

$$\begin{aligned} \text{Model 1 : } \quad X &= \sum_{i=1}^{d_Z} \alpha_i Z_i + c\varepsilon_b + \varepsilon_1, & Y &= \sum_{i=1}^{d_Z} \beta_i Z_i + c\varepsilon_b + \varepsilon_2, \\ \text{Model 2 : } \quad X &= g_1\left(\sum_{i=1}^{d_Z} Z_i + c\varepsilon_b + \varepsilon_1\right), & Y &= g_2\left(\sum_{i=1}^{d_Z} Z_i + c\varepsilon_b + \varepsilon_2\right), \end{aligned}$$

where the $Z = (Z_1, \dots, Z_{d_Z})$, ε_1 , ε_2 and ε_b are independent standard Gaussian. The coefficients $\alpha_i, \beta_i \sim U(-0.5/d_Z, 0.5/d_Z)$ follows a uniform distribution and $g_1(\cdot)$ and $g_2(\cdot)$ are uniformly chosen from $\{(\cdot), (\cdot)^2, (\cdot)^3, \tanh(\cdot), \exp(-\|\cdot\|_2)\}$. We consider (a) $H_0 : X \perp\!\!\!\perp Y \mid Z$ with $c = 0$ and (b) $H_1 : X \not\perp\!\!\!\perp Y \mid Z$ with $c = 1$.

In the following simulations, we study the test performance on different sample

sizes and dimensions of Z . The sample sizes n varies from $\{100, 200, 400, 600, 800\}$ with fixed dimensions of $d_Z = 1$ and $d_Z = 10$. The dimensions d_Z varies from $\{1, 2, 5, 10, 20\}$ with a fixed sample size of $n = 400$. We also study the effect of the cluster number M in our proposed method. The significance levels are set to be $\alpha = 0.05$ in all simulations. Evaluations of Type-I error rate, Type-II error rate, and mean runtimes are reported over 100 replications. Type-I error rate is the false rejection percentage when the underlying truth is $H_0 : X \perp\!\!\!\perp Y \mid Z$ with $c = 0$, and Type-II error rate is the false acceptance percentage when the underlying truth is $H_1 : X \not\perp\!\!\!\perp Y \mid Z$ with $c = 1$. Runtime is the average time to perform one test.

4.5.1 Hyperparameters setting

The choice of hyperparameters affects the results. For KCIT, RCIT and RCoT, the bandwidths in Gaussian kernels are set to be the squared median Euclidean distance between (X, Y) using all the pairs (or the first 500 pairs if $n > 500$) double the conditioning set size, which is recommended in [46]. CMiknn has two hyperparameters: the neighbor size $k_{\text{CMI}} = 0.1n$ in finding the estimator of the CMI, and the $k_{\text{perm}} = 5$ in the permutation, respectively. The permutation in CMiknn is repeated for 1000 times as default [48].

In our proposed methods, the bandwidths are set to be the squared median Euclidean distance between (X, Y) in each local cluster. The number of clusters M is set to be $\lceil n/50 \rceil$ when $n \leq 200$ and $\lceil n/80 \rceil$ when $n > 200$, where $\lceil x \rceil$ takes the least integer that is bigger than or equal to x . On average, each cluster has 50 samples when $n \leq 200$ and 80 samples otherwise. The local bootstrap is repeated for 1000 times.

4.5.2 When Z is low-dimensional

We first examine the performance when Z is generated independently from a standard Gaussian distribution. The sample size n changes from 100 to 800. Simulation results on linear model 1 and non-linear model 2 are reported in Figure 4.2 and Figure 4.3, respectively. Both Type-I and Type-II error rate are reported. We only report runtime in Figure 4.3 because it is not affected the models.

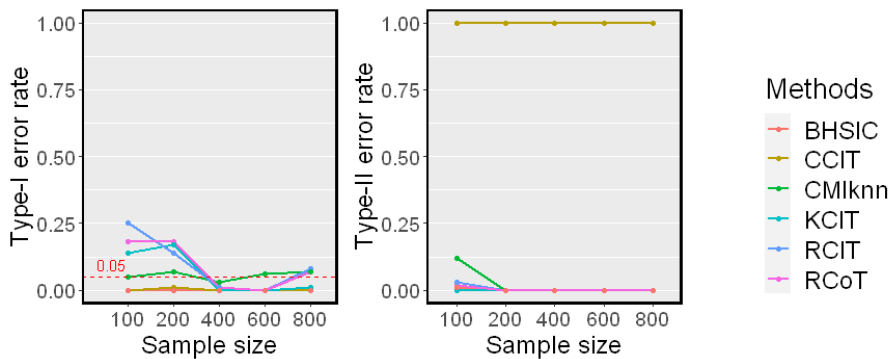


Figure 4.2: Simulation results on linear model 1 ($d_Z = 1$). The significant level is $\alpha = 0.05$. Type-I error rates, Type-II error rates are reported.

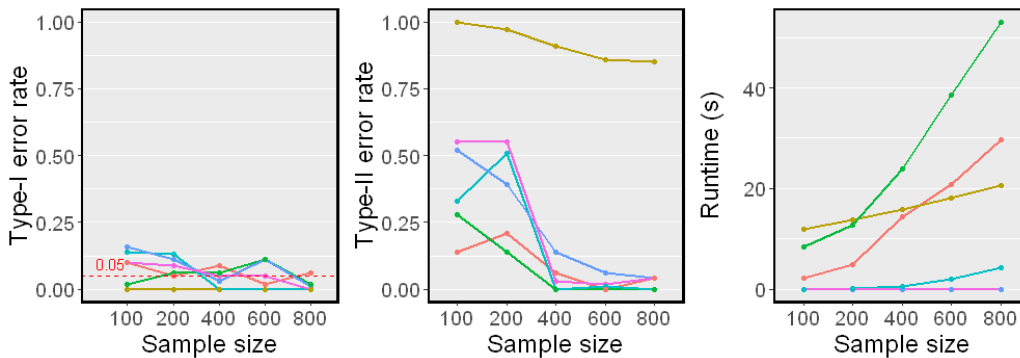


Figure 4.3: Simulation results on non-linear model 2 ($d_Z = 1$). The significant level is $\alpha = 0.05$. Type-I error rates, Type-II error rates and mean runtimes are reported.

The linear model 1 setting with a single conditioning variable Z is a simple case. All methods have controlled Type-I error rates around $\alpha = 0.05$ and almost zero

Type-II error rates, except for CCIT. In our experiments, the performance of CCIT is constantly among the worst. The data split procedure in CCIT seems to reduce the power of the test when the sample size is small. In the non-linear model 2 setting, all methods have controlled Type-I error rates around $\alpha = 0.05$. However, it shows that the proposed method and CMiknn have better powers against others when the sample size n is smaller. It matches the result in [48] that CMiknn performs well with low-dimensional conditioning set Z . When the sample size n is larger than 400, most methods have relatively low Type-II error rates. As for the runtime, the proposed method is less efficient compared to KCIT, RCIT and RCoT, which are asymptotic distribution-based methods. Though BHSIC and CMiknn are slower, the sampling procedure can readily be parallelized.

4.5.3 When Z is high-dimensional

We next examine the performance when Z is a set of 10 variables, and each variable in conditioning set Z is generated independently from a standard Gaussian distribution. The sample size n changes from 100 to 800. Simulation results on linear model 1 and non-linear model 2 are reported in Figure 4.4 and Figure 4.5, respectively.

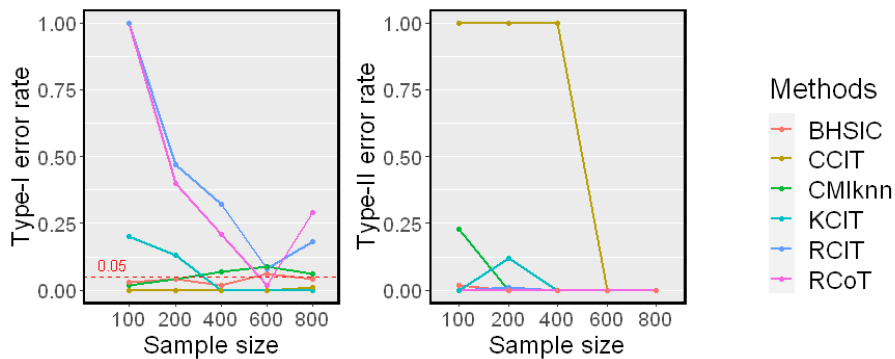


Figure 4.4: Simulation results on linear model 1 ($d_Z = 10$). The significant level $\alpha = 0.05$. Type-I error rates, Type-II error rates are reported.

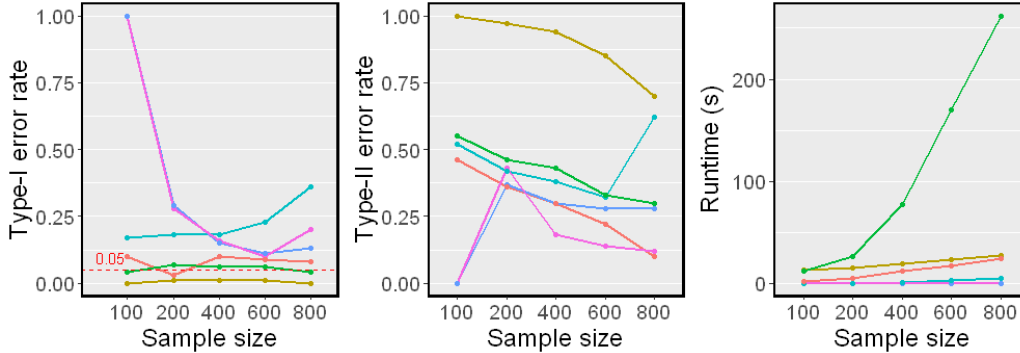


Figure 4.5: Simulation results on non-linear model 2 ($d_Z = 10$). The significant level $\alpha = 0.05$. Type-I error rates, Type-II error rates and mean runtimes are reported.

In both linear and non-linear settings, RCIT and RCoT fail and have high Type-I error rates. RCIT and RCoT approximate KCIT by using random Fourier features and are designed for large-scale datasets. Though they are more scalable than KCIT, their performances are poor when the sample size is relatively small. KCIT, CMiknn, and BHSIC perform well in the linear model setting. In the non-linear model setting, KCIT shows greater Type I error rates and Type II error rates because high-dimensional Z leads to a less accurate estimation of asymptotic distribution. We notice that BHSIC shows a higher power compared with other methods. As we expected, it is beneficial to avoid evaluating the high-dimensional Z directly, which makes the method more robust.

4.5.4 When d_Z changes

Now we evaluate the performance of the methods when the dimension of Z varies. We keep the sample size n fixed at 400 and change the dimension of Z from 1 to 20 for non-linear model 2. The results are presented in Figure 4.6. Our proposed method, BHSIC, shows good performance even with the increase in the dimension of Z , and exhibits higher power compared to other methods. It is worth noting

that the dimension of Z does not impact the runtimes since Z is only used once in the k-means algorithm, which is consistent with our computational complexity analysis.

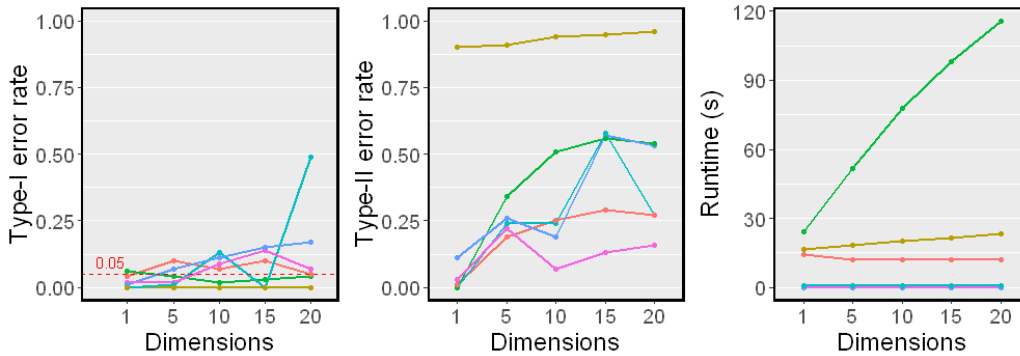


Figure 4.6: Simulation results on different dimensions of Z ($d_Z = 1, 5, 10, 15, 20$). The sample size $n = 400$ is fixed. The significant level $\alpha = 0.05$. Type-I error rates, Type-II error rates and mean runtimes are reported.

4.5.5 Effect on cluster number M

Now We study the effect on the cluster number M . We fix the sample size $n = 400$ on non-linear model 2 and change M from 2 to 20. We examine both low dimensional ($d_Z = 1$) and high dimensional ($d_Z = 10$) cases. The results are shown in Figure 4.7.

We observe that the Type I error rates are well controlled when $d_Z = 1$ and $d_Z = 10$. However, as the number of clusters M increases beyond 10, the Type-II error rate increases while the runtimes decrease. This is because when the sample points are divided into more clusters, each cluster has fewer points, leading to less accurate estimation of each local HSIC value. On the other hand, as the number of clusters increases, the computational cost of the proposed test decreases since each cluster has fewer samples and the test statistic becomes easy to calculated. The choice of clustering algorithm also affects the number of samples in each

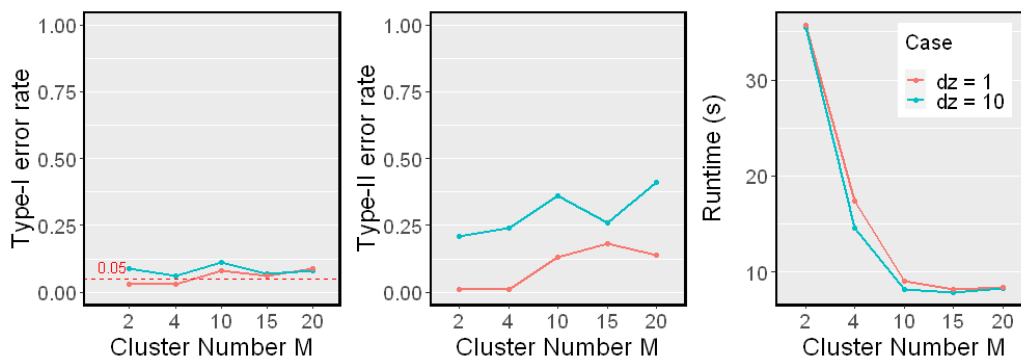


Figure 4.7: Simulation results on a different cluster number M . The sample size $n = 400$ is fixed. Results on different dimensionality of Z are reported ($d_Z = 1$, red line; $d_Z = 10$, blue line). The significant level $\alpha = 0.05$. Type-I error rates, Type-II error rates and mean runtimes are reported.

cluster. In this experiment, we used naive k-means.

4.6 Summary

In this chapter, a novel CI test is proposed. The CI test includes a new test statistic and a local bootstrap method to generate samples from the null hypothesis. In the proposed test procedure, we perform clustering method to avoid directly evaluating the high-dimensional conditioning set Z . In particular, we use the clustering result and combine several local dependence measures as a measure of conditional dependence. Consequently, the problems caused by a high-dimensional Z can be suppressed. The experimental results show that our method is robust and performs well against the growth of the dimension of the conditioning set Z .

5 | Conclusions and future work

In this chapter, we provide a summary of conclusions and a discussion about possible directions for the follow-up research.

5.1 Conclusions

Our work in Chapter 3 focuses on convex clustering, we have proposed a novel L_1 convex clustering method C-PAINT, which is highly efficient and makes it possible to visualize clusterpaths of large-scale datasets. Our proposed procedure is based on a key observation that the penalty term in L_1 convex clustering with identical weights can be reformulated into a chain-structured weighted fused Lasso. Then, we applied a refined dynamic programming optimization procedure to solve the reformulated problem. Eventually, we proposed C-PAINT algorithm to visualize the whole clusterpath. In the experiments, we demonstrate that C-PAINT outperforms existing methods including ADMM, AMA, FLSA, CARP and FUSION method in terms of efficiency while obtaining the exact solution. Our proposed method overcomes the computational challenges posed by this special case of identical weights. The runtime of C-PAINT grows almost linearly with respect to sample size n , dimension of data point p , and length of lambda sequence K , which largely expanded the application scope of convex clustering.

Our work in Chapter 4 focuses on conditional independence test, we have proposed a robust CI test that can handle high-dimensional conditioning sets Z . Our proposed CI test has two main components: a novel test statistic and a local bootstrap to sample from H_0 . For the test statistic, first, we perform k -means clustering on Z to avoid directly evaluating high-dimensional Z . Next, we measure the unconditional dependence measure HSIC for all pairs of (X, Y) in each cluster and

combine them into a single test statistic. In order to calculate the p-value, we introduce a local bootstrap sampling method to approximate the distribution under the null hypothesis. In the simulation study, we have compared our proposed method to several representative CI testing methods, including KCIT, CMiknn, RCIT, RCoT and CCIT. We have considered both a linear model and a non-linear model, with both low-dimensional and high-dimensional conditioning set Z . By combining the test statistic and local bootstrap, our proposed CI test, BHSIC, not only showed competitive performance in the low-dimensional setting but also demonstrated higher power and robust performance in cases with high-dimensional Z .

5.2 Discussion

Finally, we point out the limitations of the current works and discuss the possible directions for further improvement.

On convex clustering, though C-PAINT is highly efficient, it has two main restrictions. The first limitation is that the structure of weight in C-PAINT method is restricted to the identical weight case. However, a proper weight has a crucial effect on the performance of convex clustering, and in some scenarios, identical weights may be an infeasible choice [41]. On the other hand, though C-PAINT grows linearly as the number of λ increases when we want to obtain a more precise clusterpath or a complete dendrogram, we have to repeatedly optimize with a large number of parameters λ , which is computationally expensive.

On CI test, the proposed method is an exploration that combines clustering techniques in CI test. However, clustering results in different sizes of clusters. Thus, it is meaningful to improve the current test statistic, which is a simple mean of cluster-wise dependence measurements. Moreover, we may replace HSIC with other dependence measures or conditional dependence measures.

6 | Appendix

6.1 An example $N = 3$

Consider the case $n = 3$, given $x_1 \leq x_2 \leq x_3$, we want to minimize the following problem to obtain $\hat{a}_1 \leq \hat{a}_2 \leq \hat{a}_3$.

$$(\hat{a}_1, \hat{a}_2, \hat{a}_3) = \arg \min_{(a_1, a_2, a_3)} \left\{ \frac{1}{2} [(x_1 - a_1)^2 + (x_2 - a_2)^2 + (x_3 - a_3)^2] + \lambda_1 |a_1 - a_2| + \lambda_2 |a_2 - a_3| \right\}$$

Suppose \hat{a}_2 is known, by definition \hat{a}_1 is equal to:

$$\begin{aligned} \hat{a}_1 &= \arg \min_b \frac{1}{2} (x_1 - b)^2 + \lambda_1 |b - \hat{a}_2| \\ &= \arg \min_b h_1(b) + \lambda_1 |b - \hat{a}_2| \end{aligned}$$

Since the b here represents \hat{a}_1 and is always smaller than \hat{a}_2 . We only need to consider two cases:

- (1) $b < \hat{a}_2$. At that case, by KKT condition it is easy to find $\hat{a}_1 = U_1$.
- (2) $b = \hat{a}_2$. In other words, $\hat{a}_1 = \hat{a}_2$.

From that we get

$$\hat{a}_1 = \arg \min_b h_1(b) + \lambda_1 |b - \hat{a}_2| = \min(\hat{a}_2, U_1)$$

Similarly for \hat{a}_2 , we suppose \hat{a}_3 is known:

$$\begin{aligned}\hat{a}_2 &= \arg \min_b \left[\frac{1}{2} \{ (x_1 - \phi_1(b))^2 + (x_2 - b)^2 \} + \lambda_1 |\phi_1(b) - b| + \lambda_2 |b - \hat{a}_3| \right] \\ &= \arg \min_b h_2(b) + \lambda_2 |b - \hat{a}_3| = \min(\hat{a}_3, U_2)\end{aligned}$$

Next we need to find U_1, U_2 and \hat{a}_3 , with whom \hat{a}_1 and \hat{a}_2 can be obtained immediately. We solve it in the following order:

$$U_1 \rightarrow U_2 \rightarrow \hat{a}_3 \rightarrow \hat{a}_2 \rightarrow \hat{a}_1.$$

For U_1 , it is straightforward that $U_1 = x_1 + \lambda_1$. Then for U_2 , the U_2 satisfies $g_2(U_2) = \lambda_2$, and $g_2(b)$ is a continuous piecewise linear function shown in the figure 6.1.

$$\begin{aligned}g_2(b) &= g_1(b)\mathbf{I}[b \leq U_1] + \lambda_1\mathbf{I}[b > U_1] + (b - x_2) \\ &= (b - x_1)\mathbf{I}[b \leq U_1] + \lambda_1\mathbf{I}[b > U_1] + (b - x_2)\end{aligned}$$

Because g_2 is composed of two lines, the key is to locate which line is (U_2, λ_2) on. According to the algorithm 2, we first search from the right by assuming the (U_2, λ_2) is on the right line, and get $\beta = \lambda_2 - \lambda_1 + x_2$ which makes (β, λ_2) the intersection point of $y = \lambda_2$ with the right line. Next we compare the β with U_1 in figure 6.1. If $\beta \geq U_1$, (U_2, λ_2) is indeed on the right part of the line, then we have $U_2 = \beta$; otherwise we update the slope and intercept to be those of the left line and let $U_2 = \beta_{\text{new}} = \lambda_2 + (x_1 + x_2)/2$.

This search process is efficient enough but is still not linear. In order to make the search more efficient, some care needs to be taken.

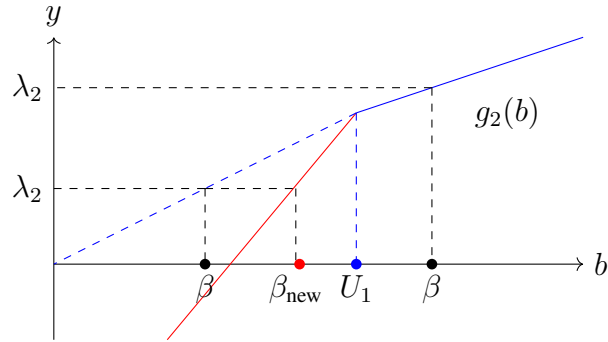


Figure 6.1: An image of $g_2(b)$: the solid line is an example of $g_2(b)$ with a branch point U_1 . The search starts to find the β first and if β does not qualify $\beta > U_1$, then we go to find the β_{new} .

Here we illustrate the erase step in line 11 of Algorithm 2. We consider the case when $U_2 < U_1$. We want to find \hat{a}_3 and search from the right part of the g_3 function. The only difference between searching U_2 and \hat{a}_3 is we let β satisfy $g_3(\beta) = 0$. We first compare the β with U_2 , and if $\beta < U_2$, the comparison between β and U_1 is no longer necessary because we already know $U_2 < U_1$. Thus we can delete the U_1 after obtaining a U_2 that is smaller than U_1 . In the end, all the U_i can be deleted at most once: once a $U_j, j > i$ is found such that $U_j < U_i$, the former U_i can be deleted immediately and never used again. By doing this, the DP algorithm becomes much more efficient and finally takes linear time.

6.2 Simulation details

Both standard errors and means over 30 replications are reported in the following table. From the table, we observed that C-PAINT is more efficient than FUSION as the sample size n grows.

Sample size n	100	500	1000	5000	10000	50000
CARP11	0.32(2.7e-3)	232.5(69)	*	*	*	*
FLSA	0.04(5.2e-4)	3.4(4.8e-2)	35.6(0.4)	*	*	*
ADMM	0.07(9.3e-4)	4.9(4.8e-2)	50.8(20)	*	*	*
AMA	0.07(1.1e-3)	3.3(2.7e-2)	42.7(15)	*	*	*
FUSION	5.0e-4(9.2e-5)	4.0e-3(1.1e-4)	1.2e-2(2.0e-4)	3.3e-1(3.1e-3)	1.1(7.4e-3)	27.7(2.8e-2)
C-PAINT	9.7e-4(2.6e-4)	3.8e-3(1.2e-4)	8.9e-3(1.5e-4)	1.5e-1(7.5e-3)	0.41(1.1e-2)	5.6(4.2e-2)

Table 6.1: Run times Comparison. The means and standard errors of each method over 30 replications are reported. Here * means we cannot obtain the solutions within a reasonable time.

List of publications

- Bingyuan Zhang, Jie Chen, and Yoshikazu Terada. Dynamic visualization for L_1 fusion convex clustering in near-linear time. Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence, PMLR 161, 515-524, 2021. <https://proceedings.mlr.press/v161/zhang21d.html>.
- Bingyuan Zhang and Joe Suzuki. Extending Hilbert–Schmidt Independence Criterion for Testing Conditional Independence. Entropy 2023. 25(3), 425. <https://doi.org/10.3390/e25030425>.
- Bingyuan Zhang and Yoshikazu Terada. (2023+) Tree-Guided L_1 -Convex Clustering. In submission.

Acknowledgements

First, I would like to thank my advisor, Professor Joe Suzuki, for his unwavering support and generous help. Joe's positive attitude, and encouragement have not only affected me but inspired me. I feel fortunate to have such a wonderful mentor, and thanks to him, the journey has been a joyful and carefree experience.

I would also like to thank Professor Yoshikazu Terada. Collaborating with him has been a valuable experience, and I am grateful for the many insightful advice he provided. I hope our collaborative relationship will continue in the future.

I want to express my gratitude to my friends and colleagues. Since coming to Japan, I have been blessed to stay in touch with my friends from afar and have wonderful friendships in Joe's lab. I am thankful for the helpful staff in the administrative office, who have made my life much easier.

Last, I would like to thank my family, who have supported me throughout this journey. I feel warm whenever I think of or talk to them. I want to thank my wife, Shinuo Wang, for the precious memories we have shared.

References

- [1] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.
- [2] 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.
- [3] Holger Hoefling. A path algorithm for the fused lasso signal approximator. *Journal of Computational and Graphical Statistics*, 19(4):984–1006, 2010.
- [4] Ryan J Tibshirani, Jonathan Taylor, et al. The solution path of the generalized lasso. *The Annals of Statistics*, 39(3):1335–1371, 2011.
- [5] Patrick Danaher, Pei Wang, and Daniela M Witten. The joint graphical lasso for inverse covariance estimation across multiple classes. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 76(2):373, 2014.
- [6] Frank Dondelinger, Sach Mukherjee, and Alzheimer’s Disease Neuroimaging Initiative. The joint lasso: high-dimensional regression for group structured data. *Biostatistics*, 21(2):219–235, 2020.
- [7] Jerome Friedman, Trevor Hastie, Holger Höfling, and Robert Tibshirani. Pathwise coordinate optimization. *The Annals of Applied Statistics*, 1(2): 302 – 332, 2007.
- [8] Oscar Hernan Madrid Padilla, James Sharpnack, and James G Scott. The dfs fused lasso: Linear-time denoising over general graphs. *Journal of Machine Learning Research*, 18(1):6410–6445, 2017.

-
- [9] 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.
- [10] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
- [11] Elias Kuthe and Sven Rahmann. Engineering fused lasso solvers on trees. In *18th International Symposium on Experimental Algorithms*, 2020.
- [12] Vladimir Kolmogorov, Thomas Pock, and Michal Rolinek. Total variation on a tree. *SIAM Journal on Imaging Sciences*, 9(2):605–636, 2016.
- [13] Peter Spirtes, Clark Glymour, and Ritchard Acheines. *Causation, Prediction, and Search*. The MIT Press, 2000.
- [14] A. J. Lawrance. On conditional and partial correlation. *The American Statistician*, 30(3):146–149, 1976.
- [15] Joe Suzuki. *Kernel Methods for Machine Learning with R*. Springer, 2022.
- [16] Joe Suzuki. *Kernel Methods for Machine Learning with Python*. Springer, 2022.
- [17] Arthur Gretton, Kenji Fukumizu, Choon Teo, Le Song, Bernhard Schölkopf, and Alex Smola. A kernel statistical test of independence. In *Advances in Neural Information Processing Systems*, pages 585–592, 2007.
- [18] Kun Zhang, Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. Kernel-based conditional independence test and application in causal discovery. In *Uncertainty in Artificial Intelligence*, page 804–813, 2011.
- [19] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf,

-
- and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(1):723–773, 2012.
- [20] Chun Li and Xiaodan Fan. On nonparametric conditional independence tests for continuous variables. *Wiley Interdisciplinary Reviews: Computational Statistics*, 12, 2019.
- [21] Toby Dylan Hocking, Armand Joulin, Francis Bach, and Jean-Philippe Vert. Clusterpath an algorithm for clustering using convex fusion penalties. In *International Conference on Machine Learning*, page 745–752, 2011.
- [22] Fredrik Lindsten, Henrik Ohlsson, and Lennart Ljung. Clustering using sum-of-norms regularization: With application to particle filter output computation. In *IEEE Statistical Signal Processing Workshop*, pages 201–204, 2011.
- [23] 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.
- [24] Eric C Chi and Kenneth Lange. Splitting methods for convex clustering. *Journal of Computational and Graphical Statistics*, 24(4):994–1013, 2015.
- [25] 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.
- [26] Peter Radchenko and Gourab Mukherjee. Convex clustering via l_1 fusion penalization. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 79(5):1527–1546, 2017.
- [27] Eric C Chi, Genevera I Allen, and Richard G Baraniuk. Convex biclustering. *Biometrics*, 73(1):10–19, 2017.
-

-
- [28] Binhuan Wang, Yilong Zhang, Will Wei Sun, and Yixin Fang. Sparse convex clustering. *Journal of Computational and Graphical Statistics*, 27(2):393–403, 2018.
- [29] Qi Wang, Pinghua Gong, Shiyu Chang, Thomas S Huang, and Jiayu Zhou. Robust convex clustering analysis. In *International Conference on Data Mining (ICDM)*, pages 1263–1268. IEEE, 2016.
- [30] Kean Ming Tan and Daniela Witten. Statistical properties of convex clustering. *Electronic journal of statistics*, 9(2):2324–2347, 2015.
- [31] Changbo Zhu, Huan Xu, Chenlei Leng, and Shuicheng Yan. Convex optimization procedure for clustering: Theoretical revisit. In *Advances in Neural Information Processing Systems*, pages 1619–1627, 2014.
- [32] 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.
- [33] Julien Chiquet, Pierre Gutierrez, and Guillem Rigail. Fast tree inference with weighted fusion penalties. *Journal of Computational and Graphical Statistics*, 26(1):205–216, 2017.
- [34] 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.
- [35] 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.
-

-
- [36] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.
- [37] Tim Stuart, Andrew Butler, Paul Hoffman, Christoph Hafemeister, Efthymia Papalexi, William M Mauck III, Yuhao Hao, Marlon Stoeckius, Peter Smibert, and Rahul Satija. Comprehensive integration of single-cell data. *Cell*, 177(7):1888–1902, 2019.
- [38] Juan G Colonna, Marco Cristo, and Eduardo F Nakamura. A distributed approach for classifying anuran species based on their calls. In *International Conference on Pattern Recognition*, pages 1242–1247. IEEE, 2014.
- [39] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.
- [40] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. *arXiv preprint arXiv 1812.01718*, 2018.
- [41] Defeng Sun, Kim-Chuan Toh, and Yancheng Yuan. Convex clustering: Model, theoretical guarantee and efficient algorithm. *Journal of Machine Learning Research*, 22(9):1–32, 2021.
- [42] Judea Pearl. *Models, Reasoning and Inference*. Cambridge University Press, 2000.
- [43] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, 2009.
- [44] Wicher Pieter Bergsma. *Testing conditional independence for continuous random variables*. Citeseer, 2004.

-
- [45] Tzee-Ming Huang. Testing conditional independence using maximal non-linear conditional correlation. *The Annals of Statistics*, 38(4):2047–2091, 2010.
- [46] Eric V. Strobl, Kun Zhang, and Shyam Visweswaran. Approximate kernel-based conditional independence tests for fast non-parametric causal discovery. *Journal of Causal Inference*, 7(1), 2019.
- [47] Gary Doran, Krikamol Muandet, Kun Zhang, and Bernhard Schölkopf. A permutation-based kernel conditional independence test. In *Uncertainty in Artificial Intelligence*, pages 132–141, 2014.
- [48] Jakob Runge. Conditional independence testing based on a nearest-neighbor estimator of conditional mutual information. In *International Conference on Artificial Intelligence and Statistics*, pages 938–947. PMLR, 2018.
- [49] Kenji Fukumizu, Arthur Gretton, Xiaohai Sun, and Bernhard Schölkopf. Kernel measures of conditional dependence. In *Advances in Neural Information Processing Systems*, 2008.
- [50] Sheng G Shi. Local bootstrap. *Annals of the Institute of Statistical Mathematics*, 43(4):667–676, 1991.
- [51] Zhen Huang, Nabarun Deb, and Bodhisattva Sen. Kernel partial correlation coefficient – a measure of conditional dependence. *Journal of Machine Learning Research*, 23(216):1–58, 2022.
- [52] Rajen D. Shah and Jonas Peters. The hardness of conditional independence testing and the generalised covariance measure. *The Annals of Statistics*, 48(3), 2020.
- [53] Rajat Sen, Ananda Theertha Suresh, Karthikeyan Shanmugam, Alexandros G Dimakis, and Sanjay Shakkottai. Model-powered conditional in-

-
- dependence test. In *Advances in Neural Information Processing Systems*, 2017.
- [54] Alexis Bellot and Mihaela van der Schaar. Conditional independence testing using generative adversarial networks. In *Advances in Neural Information Processing Systems*, 2019.
- [55] Kartik Ahuja, Prasanna Sattigeri, Karthikeyan Shanmugam, Dennis Wei, Karthikeyan Natesan Ramamurthy, and Murat Kocaoglu. Conditionally independent data generation. In *Uncertainty in Artificial Intelligence*, pages 2050–2060. PMLR, 2021.
- [56] Chengchun Shi, Tianlin Xu, Wicher Bergsma, and Lexin Li. Double generative adversarial networks for conditional independence testing. *Journal Machine Learning Research*, 22:285–1, 2021.
- [57] Shohei Shimizu, Patrik O Hoyer, Aapo Hyvärinen, Antti Kerminen, and Michael Jordan. A linear non-gaussian acyclic model for causal discovery. *Journal of Machine Learning Research*, 7(10), 2006.
- [58] Kun Zhang and Aapo Hyvarinen. On the identifiability of the post-nonlinear causal model. *arXiv preprint arXiv:1205.2599*, 2012.
- [59] Hao Zhang, Shuigeng Zhou, Kun Zhang, and Jihong Guan. Causal discovery using regression-based conditional independence tests. In *AAAI Conference on Artificial Intelligence*, pages 1250–1256, 2017.
- [60] Qinyi Zhang, Sarah Filippi, Seth Flaxman, and Dino Sejdinovic. Feature-to-feature regression for a two-step conditional independence test. In *Uncertainty in Artificial Intelligence*, 2017.
- [61] Hao Zhang, Kun Zhang, Shuigeng Zhou, Jihong Guan, and Ji Zhang. Testing

-
- independence between linear combinations for causal discovery. In *AAAI Conference on Artificial Intelligence*, pages 6538–6546, 2021.
- [62] Dimitris Margaritis. Distribution-free learning of bayesian network structure in continuous domains. In *AAAI Conference on Artificial Intelligence*, pages 825 – 830, 2005.
- [63] Elizbar A Nadaraya. On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142, 1964.
- [64] Lei Han and Yu Zhang. Reduction techniques for graph-based convex clustering. In *AAAI Conference on Artificial Intelligence*, pages 1645–1651, 2016.
- [65] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.
- [66] Thomas B Berrett, Yi Wang, Rina Foygel Barber, and Richard J Samworth. The conditional permutation test for independence while controlling for confounders. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 82(1):175–197, 2020.
- [67] Bingyuan Zhang, Jie Chen, and Yoshikazu Terada. Dynamic visualization for 11 fusion convex clustering in near-linear time. In *Uncertainty in Artificial Intelligence*, pages 515–524. PMLR, 2021.
- [68] Bingyuan Zhang and Joe Suzuki. Extending hilbert–schmidt independence criterion for testing conditional independence. *Entropy*, 25(3):425, 2023.