A Two-stage Mixed-integer Programming Approach for Portfolio Construction with K-means Clustering Technique (25)

by

Kwan Chi Yeung, Lam Ming Mei, and Chung Hon Yiu

Advised by

Professor Xin Wang

A Final Year Report Submitted to

The Hong Kong University of Science and Technology

in Partial Fulfillment of the Requirements for

the Degree of Bachelor of Engineering

in Industrial Engineering and Decision Analytics

April 2024, Hong Kong

Authorization

We hereby declare that we are the sole author of the report. We authorize the Hong Kong University of Science and Technology to lend this report to other institutions or individuals for the purpose of scholarly research. We further authorize the Hong Kong University of Science and Technology to reproduce the report by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Acknowledgement

We would like to express our deep gratitude to Professor Xin Wang for his valuable advice and support which gives us a clear direction for our project when we are encountering difficulties during our research. His constant encouragement and constructive feedback have truly made a significant impact on our work. This work was also supported by the Department of Industrial Engineering and Decision Analytics, The Hong Kong University of Science and Technology.

Abstract

This project aims to establish a two-stage mixed-integer programming approach for portfolio construction. It utilizes the k-means clustering technique as a means to avoid selecting too many securities that process very similar characteristics. Additionally, we consider those financial variables that are not included in the modern portfolio theory such as liquidity and transaction cost to overcome its related limitations. For instance, the high probability of formation of extreme portfolios. At last, the pros and cons of all algorithms used in this work for solving the integer programming form of the k-means clustering problem would also be discussed. We compare the effectiveness of each algorithm based on the squared distance from each data point to its cluster's centroid as well as their computational time.

Table of Content

Authorization ii

Acknowledgment iii

Abstract iv

Table of Content v

List of Figures vi

List of Tables vii

1 Introduction 1

2 Related Works in K-means Clustering 3

2.1 Lloyd's Algorithm 3

2.2 Frank-Wolfe Algorithm 4

2.3 Trace Maximization Algorithm 12

3 Methodology 16

3.1 K-means Clustering (Stage One) 16

3.2 Portfolio Optimization (Stage Two) 17

4 Implementation 20

4.1 K-means Clustering (Stage One) 20

4.2 Portfolio Optimization (Stage Two) 21

5 Algorithm Analysis 24

6 Portfolio Analysis 29

7 Conclusion 34

Reference 35

Appendix 38

List of Figures

Figure 1 Clustering Process of Lloyd’s Algorithm 4

Figure 2 Alternating Optimization for K-means Clustering 9

Figure 3 Frank-Wolfe Algorithm for Solving (19) 10

Figure 4 Frank-Wolfe Algorithm for Revising Matrices Z and Y 11

Figure 5 Clustering Process of Frank-Wolfe Algorithm for K-means Clustering 11

Figure 6 Trace Maximization Algorithm 14

Figure 7 Clustering Process of Trace Maximization Algorithm 15

Figure 8 Example of Convex Hull in the Two-dimensional Space 28

Figure 9 Portfolio Values 32

Figure 10 Monte Carlo Simulation for the Lloyd’s Algorithm Portfolio 33

Figure 11 Monte Carlo Simulation for the Frank-Wolfe Algorithm Portfolio 33

List of Tables

Table 1 List of Input Variables for K-means Clustering 17

Table 2 Average Financial Position of Publicly Listed Stocks in United States 20

Table 3 List of Parameters Applied in K-means Clustering 21

Table 4 Estimated Log-returns of Publicly Listed Stocks in United States 22

Table 5 Estimated Covariance Matrix of Publicly Listed Stocks in United States 23

Table 6 List of Parameters Applied in Portfolio Optimization 24

Table 7 Price of Publicly Listed Stocks in United States on 02-01-2020 24

Table 8 Number of Stocks in each Clusters 25

Table 9 Statistics Summary of the Optimal Objective Function Value 26

Table 10 Average Computational Time of each Algorithm 26

Table 11 Complexity of each Algorithm 27

Table 12 Asset Allocation in each Portfolio 29

Table 13 Statistics Summary of each Portfolio 31

Chapter 1
Introduction

The classical mean-variance analysis framework suggested by Markowitz (1952) to solve the portfolio selection problem has created a long-lasting impact. Many financial models are established based on this modern portfolio theory. It aims to build a portfolio that minimizes the risk of holding different assets given a certain level of expected return. Financial institutions nowadays are still following this approach as a foundation for developing other advanced portfolio construction methods due to its simplicity. However, error maximization tends to occur rapidly during the estimation of the sample covariance matrix (Michaud, 1998) which leads to the creation of unrealistic and extreme portfolios where weights are allocated to certain assets only. In other words, the traditional mean-variance assessment is very sensitive to a small change in the historical covariance matrix or mean vector. As a result, the objective of this paper is to demonstrate a two-stage mixed-integer programming structure for portfolio formation to overcome those limitations raised by the mean-variance theory.

Ji et al. (2014) concluded that another drawback of conventional mean-variance study is that the risk diversification principle would be usually violated as the optimal solution ultimately becomes an extreme portfolio. A common but complicated improvement for this situation is to apply robust optimization and regret optimization. Yet, there are still two important aspects that are omitted by both research. Bertsimas et al. (1999) argued that investors would consider liquidity and transaction costs when they are making investment decisions. In real practice, portfolio managers are required to manage the cash flows of portfolios accordingly because clients would demand to withdraw their investments suddenly. Hence, portfolios with low illiquidity and transaction costs are preferred by the majority. Our approach would therefore like to make use of the clustering technique as a way to really diversify the risk and also include the consideration of the two aforementioned components.

Aside from improving the mean-variance analysis, this project also intends to compare whether the current algorithms for solving the k-means clustering problem are efficient or not. In light of the short computational time and the ability to handle a large dataset, the Lloyd's algorithm is one of the most widely implemented methods. In spite of that, it is quite sensitive to the initial selection of centroids and therefore a theoretical algorithm is presented by Christian (2016). He employed a systematic way to recover the centroids and assign clusters alternatively. The idea was come from archetypal analysis which is also a type of clustering problem. On account of Christian's approach being hypothetical, we would like to verify if it is possible to use in actual scenarios. In addition to the above, contrast between these two processes would be covered in the discussion section.

Furthermore, researchers have been trying very hard to design a k-means clustering algorithm in which they do not have to guess the initial centroids. To put it another way, a superior algorithm should not be sensitive to the starting centers. Consequently, the trace maximization algorithm is targeted to solve such a matter. Combined with its simplicity, it seems to have a much better performance than the Lloyd's algorithm according to Nie et al. (2023). Thus, this project would validate whether their conclusion is true or not.

Chapter 2
Related Works in K-means Clustering

K-means clustering is a popular technique used to categorize a group of objects in terms of their characteristics. In the world of finance, one application is to determine the probability of a financial crisis via the classification of macroeconomic risk factors (Zhu & Liu, 2021). In addition, it may be applied in stock recommendation system for shareholders to earn a considerable profit (Fang & Chiao, 2021). An effective clustering algorithm seeks to discover the homogeneity or heterogeneity among different data points. Thus, a good clustering result should consist of both a high intra-class similarity and a low inter-class similarity.

2.1 Lloyd's Algorithm

Given there are N data points in a m-dimensional space, k-means clustering attempts to assign each data point to exactly one of the K clusters such that the sum of squared distances of each cluster is minimized. This optimization problem could be formulated mathematically as follows

$$\min \left(\sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2 \right) \tag{1}$$

where x_i is a vector representing the i-th data point and μ_k is the centroid of cluster C_k which is defined as the arithmetic mean.

Another straightforward expression is to transform it into an integer programming structure through the introduction of dummy variables. The interior point algorithm proposed by Merle et al. (1999) relies on this interpretation and the objective function becomes

$$\min \left(\sum_{k=1}^{K} \sum_{i=1}^{N} z_{ki} \|x_i - \mu_k\|^2 \right) \tag{2}$$

where z_{ki} is a binary variable that takes the value of one only if the i-th data point is a member of the k-th cluster. Furthermore, an extra constraint must be added to ensure a data point belongs to one and only one cluster. If we fix the data index i and sum over the cluster index k, then constraint (3) should hold.

$$\sum_{k=1}^{K} z_{ki} = 1 \tag{3}$$

There are indeed many efficient algorithms that could be used to solve the k-means clustering problem. Nonetheless, here we first introduce the most well-known one which is named as the Lloyd's algorithm (Rebollo-Monedero, 2013). To start with, randomly initialize K centroids where these centroids must lie within the data domain. Each data point is assigned to its nearest cluster in the next step based on the squared Euclidean distance between the data point and the centroid. This is followed by computing the new arithmetic mean of each cluster. At last, repeat the last two steps until all centroids converge. Some may also run this entire process multiple times with different random initial centroids and pick the best clustering afterward to increase the accuracy.

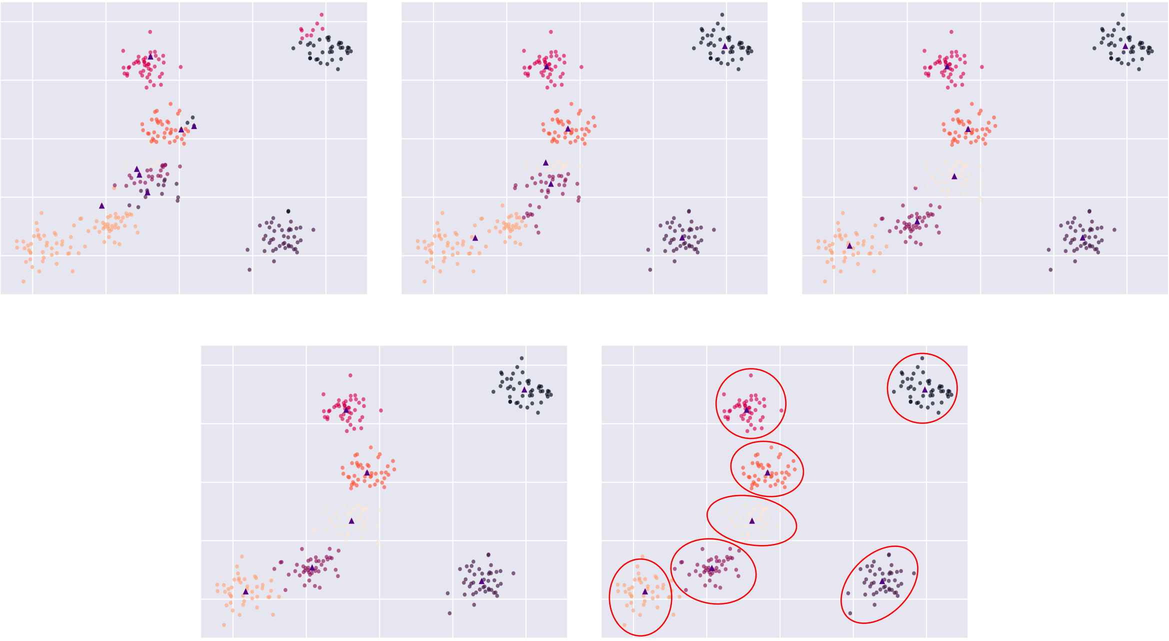

Figure 1. Clustering Process of Lloyd's Algorithm

Above figures show the clustering process of this Lloyd's algorithm and the centroid of each cluster is labelled by the triangle symbol. Its advantages and disadvantages would be further discussed in a later section.

2.2 Alternating Frank-Wolfe Algorithm

The second algorithm we would illustrate in this section is so-called the alternating Frank-Wolfe technique (Christian, 2016). In order to understand its principle, we need to prove several important properties of the aforementioned integer programming framework. Let X be a $m \times N$ matrix such that its i-th column is a m-dimensional vector which represents the i-th data point. Then the following relationship is true

$$\sum_{k=1}^{K}\sum_{i=1}^{N} z_{ki}\|x_i - \mu_k\|^2 = \|X - MZ\|^2 \tag{4}$$

where M is a $m \times K$ matrix that contains all K unique centroids and Z is a $K \times N$ binary indicator matrix in which each row must exactly sum to one.

The first step to prove this identity is to expand the left-hand side of (4) and it involves numerous complicated calculations as shown below. Christian (2016) argued that the objective function of k-means clustering should consist of three terms.

$$\sum_{k=1}^{K}\sum_{i=1}^{N} z_{ki}\|x_i - \mu_k\|^2 = \sum_{k=1}^{K}\sum_{i=1}^{N} z_{ki}\|x_i\|^2 - 2\sum_{k=1}^{K}\sum_{i=1}^{N} z_{ki}(x_i \cdot \mu_k) + \sum_{k=1}^{K}\sum_{i=1}^{N} z_{ki}\|\mu_k\|^2 \tag{5}$$

He examined the first and third components separately and rewrote each one into a trace notation as in (6) and (7) respectively

$$\sum_{k=1}^{K}\sum_{i=1}^{N} z_{ki}\|x_i\|^2 = \sum_{i=1}^{N}\|x_i\|^2 \sum_{k=1}^{K} z_{ki} = \sum_{i=1}^{N}\|x_i\|^2 = \mathrm{tr}(X^{\mathrm{T}}X) \tag{6}$$

$$\sum_{k=1}^{K}\sum_{i=1}^{N} z_{ki}\|\mu_k\|^2 = \sum_{k=1}^{K}\|\mu_k\|^2 \sum_{i=1}^{N} z_{ki} = \sum_{k=1}^{K} \mathrm{n}_k\|\mu_k\|^2 \tag{7}$$

where n_k is the total number of data points in the k-th cluster. The second term in (5) could also be simplified as in (8) and the i-th column of a matrix A is denoted by A_i from here and onward. On top of that, the expression $(A)_{a,b}$ is referring to its (a, b) element.

$$\sum_{k=1}^{K}\sum_{i=1}^{N} z_{ki}(x_i \cdot \mu_k) = \sum_{k=1}^{K}\sum_{i=1}^{N} z_{ki}\sum_{j=1}^{m}(x_{ji} \cdot \mu_{jk}) = \sum_{i=1}^{N}\sum_{j=1}^{m} x_{ji}\sum_{k=1}^{K}(\mu_{jk} \cdot z_{ki}) \tag{8}$$

$$= \sum_{i=1}^{N}\sum_{j=1}^{m} x_{ji}(MZ)_{ji} = \sum_{i=1}^{N}\sum_{j=1}^{m} (X^{\mathsf{T}})_{ij}(MZ)_{ji}$$

$$= \sum_{i=1}^{N} (X^{\mathsf{T}}MZ)_{ii} = \mathrm{tr}(X^{\mathsf{T}}MZ)$$

Next, Christian (2016) looked at the right-hand side of (4). This term is known as the squared Frobenius norm of matrix which is equal to the trace of the product of that matrix (Baksalary & Trenkler, 2014).

$$\|X - MZ\|^2 = \mathrm{tr}((X - MZ)^{\mathsf{T}}(X - MZ)) = \mathrm{tr}(X^{\mathsf{T}}X) - 2\mathrm{tr}(X^{\mathsf{T}}MZ) + \mathrm{tr}(Z^{\mathsf{T}}M^{\mathsf{T}}MZ) \tag{9}$$

$$= \mathrm{tr}(X^{\mathsf{T}}X) - 2\mathrm{tr}(X^{\mathsf{T}}MZ) + \mathrm{tr}(M^{\mathsf{T}}MZZ^{\mathsf{T}})$$

To validate whether (5) and (9) are equivalent to each other, the third expression in (9) has to be converted to guarantee it is identical to (7) since the first and second terms in (5) are the same as that in (9).

$$\mathrm{tr}(M^{\mathsf{T}}MZZ^{\mathsf{T}}) = \sum_{k=1}^{K} (M^{\mathsf{T}}MZZ^{\mathsf{T}})_{kk} = \sum_{k=1}^{K} (M^{\mathsf{T}}M)_{kk}(ZZ^{\mathsf{T}})_{kk} = \sum_{k=1}^{K} \mathrm{n}_k\|\mu_k\|^2 \tag{10}$$

Consequently, identity (4) holds and k-means clustering in fact is a matrix factorization problem. In order to minimize the objective function for a given matrix Z, Christian (2016) differentiated (9).

$$\frac{\partial}{\partial M}\|X - MZ\|^2 = \frac{\partial}{\partial M}(\mathrm{tr}(X^\mathsf{T}X) - 2\mathrm{tr}(X^\mathsf{T}MZ) + \mathrm{tr}(M^\mathsf{T}MZZ^\mathsf{T})) = -2XZ^\mathsf{T} + 2MZZ^\mathsf{T} \tag{11}$$

$$M = XZ^\mathsf{T}(\,ZZ^\mathsf{T}\,)^{-1} \tag{12}$$

Eventually, the objective function of a k-means clustering problem could be stated in terms of the data matrix X and the binary indicator matrix Z only.

$$\min \quad \left\|X - XZ^\mathsf{T}(\,ZZ^\mathsf{T}\,)^{-1}\,Z\right\|^2 \tag{13}$$

After discussing the properties of k-means clustering, we may move forward to suggest that (13) could actually be viewed as a kind of archetypal analysis. The purpose of this analysis is to find the archetypes within a data set (Eugster & Leisch, 2009). In other words, it tries to approximate the centroids of clusters by a convex combination of all data points. For this reason, Christian (2016) first added a new column stochastic matrix Y (all elements in the same column in a column stochastic matrix must sum to one) in his analysis such that M could be reformulated as in (14).

$$M = XZ^\mathsf{T}(ZZ^\mathsf{T})^{-1} = XY \tag{14}$$

It is worth pointing out that each column in Z always produces a zero entropy because of its binary nature and thus, columns in Y should also develop a very similar property which leads to

$$H(Z_i) = -\sum_{k=1}^{K} Z_{ki} \ln Z_{ki} = 0 \tag{15}$$

$$H(Y_k) = -\sum_{i=1}^{N} Y_{ik} \ln Y_{ik} \gg 0 \tag{16}$$

Information theory uses entropy to report the long-lasting behavior of a random process (Gray, 2011) and according to constraints (15) and (16), k-means clustering turns out to be a constrained quadratic optimization problem.

$$\min \quad \|X - XYZ\|^2 \tag{17}$$

$$\text{s.t.} \qquad Z \succcurlyeq 0, 1^{\mathrm{T}} Z_i = 1, H(Z_i) = 0$$

$$Y \succcurlyeq 0, 1^{\mathrm{T}} Y_k = 1, H(Y_k) \gg 0$$

Since k-means clustering is found to be a constrained version of archetypal analysis, Christian (2016) recommended that the alternating optimization idea which was originally developed for performing archetypal analysis could be applied here to compute the clustering result. Nonetheless, several adjustments have to be carried out to make the problem much easier to solve. To begin with, model (17) requires the columns of Y to be a standard simplex. A standard N-simplex is an object that is made up of $(N + 1)$ vertices (Aguadé & Richard, 1987). These vertices should form a convex set, also known as a convex hull, such that the line segment between any points in the N-simplex lies within the object. For instance, a standard basis vector with a dimension of $(N + 1)$ is a standard N-simplex.

$$\Delta^{N-1} = \{\theta \in \mathbb{R}^N \mid \theta \succcurlyeq 0 \wedge 1^{\mathrm{T}} \theta = 1\} \tag{18}$$

Another thing to note here is the concavity of the entropy function. Because we are interested in maximizing it according to (16) and would like to tackle a convex minimization problem, the column entropy therefore should be multiplied by negative one in order to make it convex. All in all, the final problem becomes

$$\min \quad \|X - XYZ\|^2 - \sum_{k=1}^{K} H(Y_k) \tag{19}$$

$$\text{s.t.} \qquad Y_k \in \Delta^{N-1}$$

Based on this ultimate formulation, an optimization strategy is to compute matrices Z and Y under the appropriate constraints alternatively.

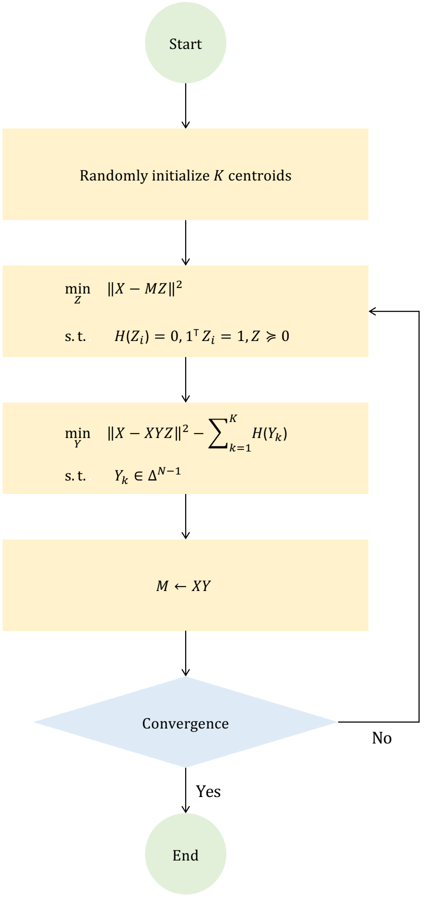
Start
Randomly initialize K centroids
$\min_{Z} \quad \|X - MZ\|^2$
s.t. $\quad H(Z_i) = 0, 1^\mathsf{T} Z_i = 1, Z \geqslant 0$
$\min_{Y} \quad \|X - XYZ\|^2 - \sum_{k=1}^{K} H(Y_k)$
s.t. $\quad Y_k \in \Delta^{N-1}$
$M \leftarrow XY$
Convergence
No
Yes
End

Figure 2. Alternating Optimization for K-means Clustering

A standard N-simplex is deemed to be a compact convex set as it is bounded by $(N + 1)$ standard basis vectors $e \in \mathbb{R}^{N+1}$ (Christian, 2020). As a result, the two subproblems written in the above figure have the same general form as stated in (20). Its optimal solution could be figured out with the use of the Frank-Wolfe algorithm where a compact convex set is indicated by S.

$$\min_{x \in S} \quad F(x) \tag{20}$$

The purpose of this algorithm is to compute $s \in S$ that minimizes $s^T \nabla F(x_t)$ which moves the current iterate x_t along a descent direction $s - x_t$ with a suitable step size. This protects the search would never leave the feasible region and moreover, Bomze et al. (2020) summarized that a zigzag pattern could be obtained if the sequence of these vertices is plotted on a graph.

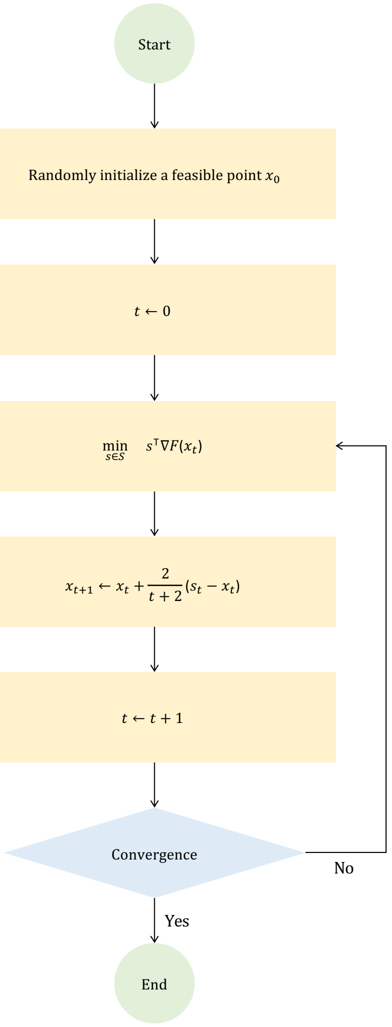
Start
Randomly initialize a feasible point x_0
$t \leftarrow 0$
$\min_{s \in S} \quad s^\top \nabla F(x_t)$
$x_{t+1} \leftarrow x_t + \frac{2}{t+2}(s_t - x_t)$
$t \leftarrow t+1$
Convergence
No
Yes
End

Figure 3. Frank-Wolfe Algorithm for Solving (19)

Regarding model (19), we may evaluate matrices Z and Y with reference to the two algorithms displayed below and the gradient matrix G with respect to each matrix is given in (21) and (22).

$$\nabla_Y = 2(X^\top XYZZ^\top - X^\top XZ^\top) - (\log Y + 1) \tag{21}$$

$$\nabla_Z = 2(M^\top MZ - M^\top X) \tag{22}$$

Owing to the fact that the compact convex set S is actually a standard simplex, knowing the optimal solution of s is equivalent to knowing the standard basis vector e_l that would minimize the scalar product of this standard basis vector and the l-th column of the gradient matrix G. This all leads to

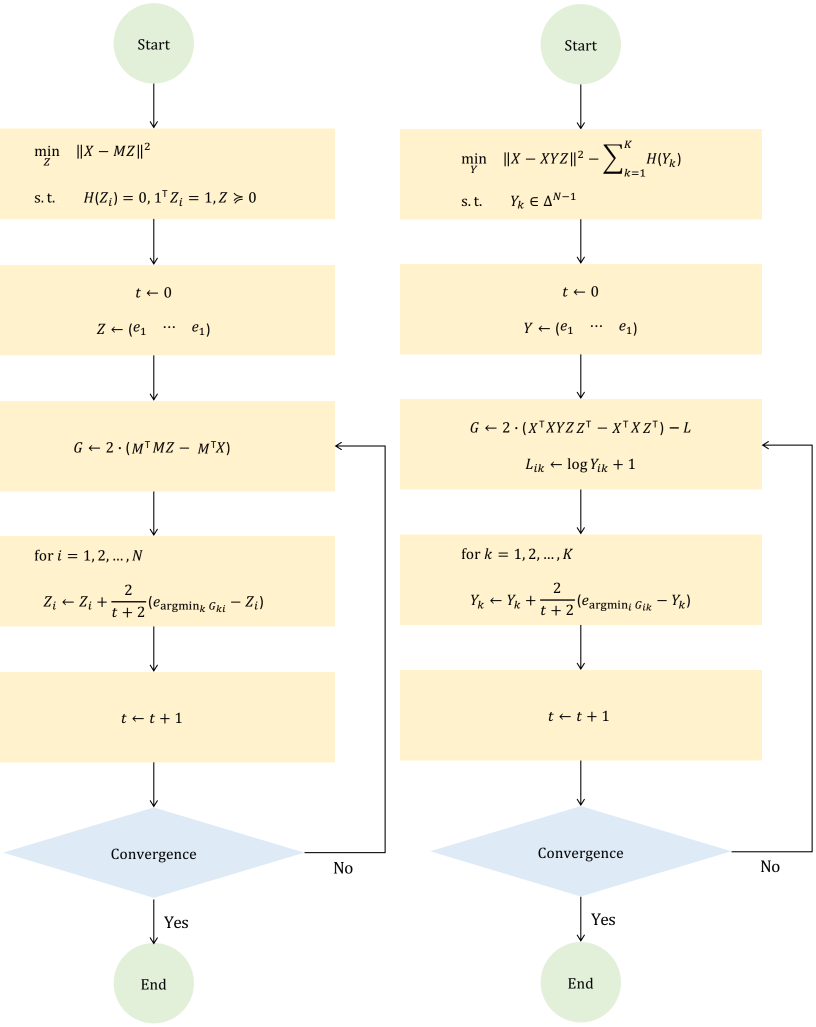
Start
$\min_Z \quad \|X - MZ\|^2$
s.t. $H(Z_i) = 0, 1^\top Z_i = 1, Z \geqslant 0$
$t \leftarrow 0$
$Z \leftarrow (e_1 \quad \cdots \quad e_1)$
$G \leftarrow 2 \cdot (M^\top MZ - M^\top X)$
for $i = 1, 2, \dots, N$
$Z_i \leftarrow Z_i + \frac{2}{t+2}(e_{\operatorname{argmin}_k G_{ki}} - Z_i)$
$t \leftarrow t + 1$
Convergence
No
Yes
End
Start
$\min_Y \quad \|X - XYZ\|^2 - \sum_{k=1}^{K} H(Y_k)$
s.t. $Y_k \in \Delta^{N-1}$
$t \leftarrow 0$
$Y \leftarrow (e_1 \quad \cdots \quad e_1)$
$G \leftarrow 2 \cdot (X^\top XYZZ^\top - X^\top XZ^\top) - L$
$L_{ik} \leftarrow \log Y_{ik} + 1$
for $k = 1, 2, \dots, K$
$Y_k \leftarrow Y_k + \frac{2}{t+2}(e_{\operatorname{argmin}_i G_{ik}} - Y_k)$
$t \leftarrow t + 1$
Convergence
No
Yes
End

Figure 4. Frank-Wolfe Algorithm for Revising Matrices Z and Y

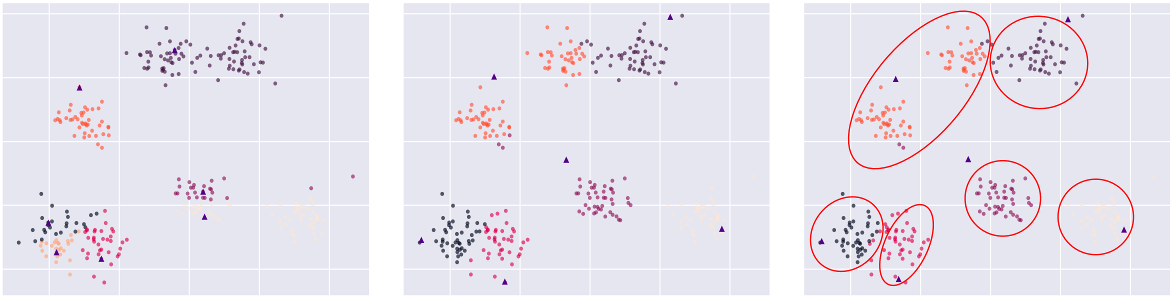

Figure 5. Clustering Process of Frank-Wolfe Algorithm for K-means Clustering

The clustering process of the Frank-Wolfe algorithm is demonstrated above. Although it does disclose a good clustering outcome, the reason why it is usually not applicable would be examined later in terms of different aspects.

2.3 Trace Maximization Algorithm

To overwhelm the performance of the Lloyd's algorithm, some may advance the centroids initialization method. In this case, all centroids are not generated arbitrarily anymore in the first step but according to a probability distribution. In the same vein, many researchers would like to solve the k-means clustering problem without having to guess the centroids. Hence, considering equalities (4) and (9) again. The objective function of the k-means clustering problem is consistent with

$$\min \quad (\mathrm{tr}(X^{\mathsf{T}}X) - 2\mathrm{tr}(X^{\mathsf{T}}MZ) + \mathrm{tr}(Z^{\mathsf{T}}M^{\mathsf{T}}MZ)) \tag{23}$$

If (12) is substituted into (23), the objective function could be rewritten as a trace maximization problem.

$$\max \quad \mathrm{tr}(ZX^{\mathsf{T}}XZ^{\mathsf{T}}(\,ZZ^{\mathsf{T}}\,)^{-1}) \tag{24}$$

The k-th diagonal element of ZZ^{T} is equal to the inner product of the k-th row vector of Z with itself. Therefore, (24) may be changed to (25) if the transpose of Z is denoted by F.

$$\max \quad \sum_{k=1}^{K} \frac{(F_k)^{\mathsf{T}}X^{\mathsf{T}}X\,F_k}{(F_k)^{\mathsf{T}}F_k} \tag{25}$$

On account of the difficulty of solving (25), a k-dimensional vector v is introduced such that it could now be expressed by

$$\max \quad \sum_{k=1}^{K} \left(2v_k\sqrt{(F_k)^{\mathsf{T}}X^{\mathsf{T}}XF_k} - (v_k)^2(F_k)^{\mathsf{T}}F_k\right) \tag{26}$$

The brand new algorithm advocated by Nie et al. (2023) to tackle the k-means clustering problem is built on this presentation. It retrieves the value of v and F alternatively before they converge. As the foremost step, randomly construct the indicator matrix and first update the elements in v by equation (28).

$$\frac{\partial}{\partial v_k}\left(2v_k\sqrt{(F_k)^{\top}X^{\top}XF_k}-(v_k)^2(F_k)^{\top}F_k\right)=2\sqrt{(F_k)^{\top}X^{\top}XF_k}-2v_k(F_k)^{\top}F_k \tag{27}$$

$$v_k=\frac{\sqrt{(F_k)^{\top}X^{\top}XF_k}}{(F_k)^{\top}F_k} \tag{28}$$

Following the revising of vector v, matrix F is restored with the aid of the repetitive re-weighted method (Nie et al., 2017). In other words, the optimal solution of (29) could be found by iteratively maximizing (30) until x converges

$$\max \quad \sum_i H_i(G_i(x))+F(x) \tag{29}$$

$$\max \quad \sum_i \operatorname{tr}\left(\left(\frac{\partial H_i(G_i(x))}{\partial G_i(x)}\right)^{\top} G_i(x)\right)+F(x) \tag{30}$$

where (29) is the general form of (26) and $H_i(\,\cdot\,)$ is convex. For our case, the repetitive function is given by (31) and some notations are also simplified here to facilitate reader's understanding. Let Q be a matrix such that its k-th column is

$$\max \quad \sum_{k=1}^{K}\left(2v_k(F_k)^{\top}\frac{X^{\top}XF_k}{\sqrt{(F_k)^{\top}X^{\top}XF_k}}-(v_k)^2(F_k)^{\top}F_k\right) \tag{31}$$

$$Q_k=\frac{\partial\sqrt{(F_k)^{\top}X^{\top}XF_k}}{\partial F_k}=\frac{X^{\top}XF_k}{\sqrt{(F_k)^{\top}X^{\top}XF_k}} \tag{32}$$

Then specify two $N \times K$ matrices A and B where $A = 2QV$ and $B = 1\tilde{V}$ while V and $\tilde{V}$ are two $K \times K$ diagonal matrices. The diagonal elements of V and $\tilde{V}$ are v_k and $(v_k)^2$ respectively. Then we have

$$\max \quad \mathrm{tr}(F^{\mathsf{T}}(A - B)) \tag{33}$$

Therefore, we may use the definition of A and B to solve (31). Given the binary nature of F^{T} and its column sum is equal to one, each row of F is a standard basis vector. Hence, what is required here is to find the position of the largest element in each row of $(A - B)$. For instance, F_{ik} would equal to one if the greatest element in the i-th row of $(A - B)$ has an index k.

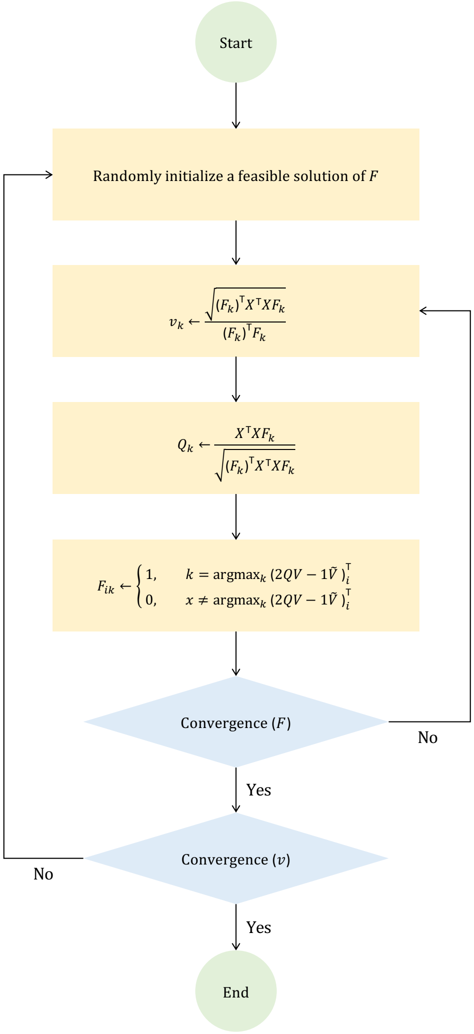
Start
Randomly initialize a feasible solution of F
$v_k \leftarrow \frac{\sqrt{(F_k)^{\mathsf{T}} X^{\mathsf{T}} X F_k}}{(F_k)^{\mathsf{T}} F_k}$
$Q_k \leftarrow \frac{X^{\mathsf{T}} X F_k}{\sqrt{(F_k)^{\mathsf{T}} X^{\mathsf{T}} X F_k}}$
$F_{ik} \leftarrow \begin{cases} 1, & k = \mathrm{argmax}_k\,(2QV - 1\tilde{V})_i^{\mathsf{T}} \\ 0, & x \neq \mathrm{argmax}_k\,(2QV - 1\tilde{V})_i^{\mathsf{T}} \end{cases}$
Convergence (F)
No
Yes
Convergence (v)
No
Yes
End

Figure 6. Trace Maximization Algorithm

To bring the algorithm to an end, the k-means clustering problem could be formulated as (26). Furthermore, (26) is solved by alternatively obtaining v and F until they are both converged. The clustering progress is again shown below.

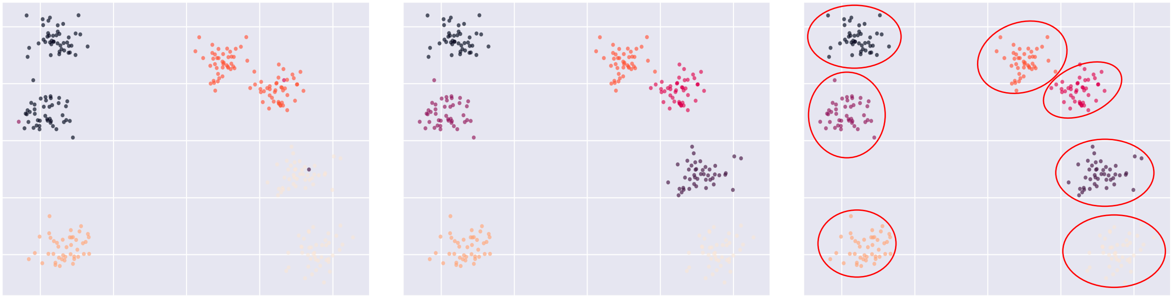

Figure 7. Clustering Process of Trace Maximization Algorithm

Chapter 3
Methodology

Evidence has been provided in many researches to reveal the disadvantages of the mean-variance analysis (Fulga, 2015) such as its relatively high possibility of getting an extreme portfolio. Therefore, only relying on this analytical tool is not adequate. The prototype designed by our team adopts a two-stage mixed-integer programming structure. Its target is to carefully pick up a limited number of assets which share dissimilar features for diversification through k-means clustering. It further takes liquidity and transaction costs into account which are two vital factors that investors may think about.

3.1 K-means Clustering (Stage One)

The diversification benefit depends not only on the number of securities but also the correlation among them. To put it another way, a portfolio is said to be more diversified than another if they have the same number of securities and each pair of security prices in the more diversified portfolio moves more differently. As a result, some may believe that fundamental analysis is a good starting point. It predicts the intrinsic value of a security by gathering financial variables from the financial statements. This assists traders in deciding whether a long or a short position should be taken and could indirectly help them to choose a proper set of uncorrelated securities which acts as a potential portfolio candidate. As a consequence of using financial variables to value an asset, another method to find uncorrelated constituents is to conduct a cluster analysis because it strives to group indistinguishable assets together. By randomly picking a security in each cluster, a partly diversified portfolio may be formed.

K-means clustering is one of the ways to group a list of items in relation to their attributes. It explores the data patterns from a large amount of unlabeled inputs. Seeing that a security could belong to multiple categories, it is difficult to use only one financial feature to summarize a security. For instance, Standard and Poor's sorts publicly traded companies into different sectors according to their business activity while they may also be separated based on their market capitalization. Therefore, unsupervised learning like k-means clustering is especially useful for securities segmentation as there are no clear definitions of clusters. Since we would like to set up an integer programming problem, the objective function and constraint are the same as (2) and (3) while the list of input variables is as follows.

Table 1. List of Input Variables for K-means Clustering

(1) Net Income	(10) Operating Revenue
(2) Inventory	(11) Cost of Goods Sold
(3) Current Assets	(12) Stockholders' Equity
(4) Total Assets	(13) Earnings before Interest, Taxes, Depreciation, and Amortization
(5) Current Liabilities	(14) Earnings before Interest and Taxes
(6) Total Debt	(15) Interest Expense
(7) Gross Profit	(16) Cash and Cash Equivalents
(8) Working Capital	(17) Dividends Paid Per Share
(9) Invested Capital	(18) Number of Dividends Paid

3.2 Portfolio Optimization (Stage Two)

By exploiting the advantage of k-means clustering, all available assets are now being split into numerous clusters. To fully diversify our portfolio, we begin with the regular mean-variance analysis (Markowitz, 1952). It offers a simple and convenient way for managers to determine the ideal weight of each asset in a portfolio regarding their individual returns and the correlation among them. To obtain the minimum variance portfolio for a given level of expected return β, it is necessary to evaluate the optimal solution of the following problem.

$$\begin{aligned} \min \quad & w^{\mathsf{T}} \sum w \qquad (34) \\ \text{s.t.} \quad & w^{\mathsf{T}} \mu \geq \beta \\ & 1^{\mathsf{T}} w = 1 \end{aligned}$$

In reality, there are many more constraints that should be included for enhancing the robustness of this study. For example, the incurred transaction costs when trading a security and the liquidity needs of investors. Hence, it is particularly helpful to alter the model to satisfy the real situation. First, investors bear the transaction costs when they are trading an asset. They estimate their total trading expenses by the bid-ask spread or the commission fees charged by the brokers or dealers. For simplicity, this model assumes that investors do not have to compensate for the bid-ask spread in which the transaction cost function could be represented by

$$T = t + \frac{c}{P} \tag{35}$$

where c is the brokerage fees per share (in dollar) and t is the taxes collected by the government (in percentage) if applicable. Together with the return constraint, this trading expenditure may be deducted from the security return. Thus, it is replaced by

$$w^{\intercal}\left(\mu - t - \frac{c}{P}\right) \geq \beta \tag{36}$$

After capturing the transaction cost function, we move on to consider the liquidity of the portfolio. Amihud (2002) presented a state-of-the-art measure of illiquidity for his cross-section and time-series study of stock excess returns. He interpreted illiquidity as

$$\frac{1}{D}\sum\nolimits_{d}^{D}\frac{|R_d|}{V_d} \tag{37}$$

where R is the daily return and V is the daily dollar trading volume. This measure reflects the response of price associated with a dollar of trading volume which may imply the liquidity premium paid by the buyers. Investors generally opt for a portfolio with a lower illiquidity ratio because they may retire their part of investments unexpectedly. To fulfil the liquidity demand, constraint (38) is added

$$w^{\intercal} I \leq L \tag{38}$$

where I is the illiquidity vector and L is the maximum illiquidity ratio of the portfolio. To completely conclude this mixed-integer programming problem, the number of assets filtered out from each cluster is not permitted to exceed the predefined value. Subsequently, binary variables are introduced here to do the job.

$$\sum_{i \in C_k} B_i \leq N_{C_k} \tag{39}$$

$$\sum_{i \in C_k} B_i \geq 1 \tag{40}$$

Furthermore, the binary variable B_i takes the value of one when asset i is chosen and zero if it is not. To make this happen, the binary vector B has to meet the inequality $B \geq w$.

Chapter 4
Implementation

The two-stage mixed-integer programming model was implemented in actual circumstance to test its effectiveness. Significant information would be supplied in this section for readers to mimic our approach.

4.1 K-means Clustering (Stage One)

For the k-means clustering problem in stage one, financial variables were used as the input and it could be solved by any of the aforementioned algorithms as elaborated in the second chapter. Below is a part of the data collected from *Yahoo Finance* which is the average financial position of each publicly listed stock in United States.

Table 2. Average Financial Position of Publicly Listed Stocks in United States (2020 – 2022)

Ticker (E+08)	Inventory	Working Capital	⋯	Total Debt
A	8.626667E+08	1.985333E+09	⋯	2.619000E+09
AA	1.927000E+09	1.936000E+09	⋯	2.118667E+09
⋮	⋮	⋮	⋱	⋮
ZYXI	1.095833E+07	5.373233E+07	⋯	2.312600E+07

These data should first be normalized before clustering is done because this could increase the training stability and reduce the computational time. As a result, we scaled all data point by the min-max standardization formula and this could be achieved by calling the *MinMaxScaler* function from *Sklearn*.

Since the idea of all concerned algorithms to solve the k-means clustering problem has already been mentioned, this part would be ended here to prevent overlap. Please refer to the appendix to check how they could be executed in Python and so as the list of parameters appeared as follows.

Table 3. List of Parameters Applied in K-means Clustering
(1) n_clusters = 7
(2) n_init = 500 (number of times the algorithm is run with different initialization setup)
(3) init_method = random (method for initialization)
(4) outer_iter = 300 (number of iteration in the outer loop)
(5) inner_iter = 100 (number of iteration in the inner loop)

4.2 Portfolio Optimization (Stage Two)

Before discussing how to deal with the discrete optimization problem in stage two, we present a reasonable way to approximate the expected return of each asset. A recommendation is to use the adjusted closing price from *Yahoo Finance*. It not only explicates for stock splits and dividends but also adheres to the Center for Research in Security Price standards. Given that the expected return incorporates both the appreciation of stock price and dividend yield, using the adjusted closing price is particularly handy when analyzing a security. On top of this, we also use an estimator that is robust to heavy tails for both the return vector and the covariance matrix in order to achieve a completely diversified portfolio. This allows the existence of outliers while the evaluation target would not vary too much (Huber & Ronchetti, 2009). One such method is to apply the minimization of the negative log-likelihood function.

$$\frac{N+\eta}{T}\sum_{t=1}^{T}\frac{x_t-\mu}{\eta+(x_t-\mu)^{\mathrm{T}}\Sigma^{-1}(x_t-\mu)}=0 \tag{41}$$

$$\frac{N+\eta}{T}\sum_{t=1}^{T}\frac{(x_t-\mu)(x_t-\mu)^{\mathrm{T}}}{\eta+(x_t-\mu)^{\mathrm{T}}\Sigma^{-1}(x_t-\mu)}=\Sigma \tag{42}$$

In other words, μ and Σ could be found by solving the simultaneous equations (41) and (42) where η is the degree of freedom and N is the dimension of μ. Unfortunately, there is no closed-form solution and it needs to be approximated by an iterative method (Ying Sun et al., 2015)

$$\mu_j = \frac{1}{w_j}\sum_{t=1}^{T}\frac{N+\eta}{\eta + (x_t - \mu_{j-1})^{\mathsf{T}}(\Sigma_{j-1})^{-1}(x_t - \mu_{j-1})}\cdot x_t \tag{43}$$

$$\Sigma_j = \frac{1}{T}\sum_{t=1}^{T}\frac{N+\eta}{\eta + (x_t - \mu_{j-1})^{\mathsf{T}}(\Sigma_{j-1})^{-1}(x_t - \mu_{j-1})} \times (x_t - \mu)(x_t - \mu)^{\mathsf{T}} \tag{44}$$

$$w_j = \sum_{t=1}^{T}\frac{N+\eta}{\eta + (x_t - \mu_{j-1})^{\mathsf{T}}(\Sigma_{j-1})^{-1}(x_t - \mu_{j-1})} \tag{45}$$

where j is the iteration number. Although the robust estimator explained here grants a rational estimation, analysts may face challenging in data collection. To paraphrase, the sample size of the daily return is not large enough compared to the number of securities such that T is much smaller than or equal to N in this case. This causes an inaccurate estimator to take place and therefore, we also offer a simple way to conduct the same task. This project deployed the second approach where the expected return could be replaced by the trimmed mean return which is the average value after deleting a portion of outliers.

Table 4. Estimated Log-returns of Publicly Listed Stocks in United States

Ticker	Log-return
A	−1.33300896E−04
AA	−3.61469797E−04
⋮	⋮
ZYXI	6.78382262E−04

For the covariance matrix, the Tyler's estimator is used according to Ying Sun et al. (2014). The reason is that this estimator does not require user to predict the mean vector at the same time and as a result, it would be less time consuming when the dataset is large (no matter is the number of securities or the number of period). Furthermore, the estimated covariance matrix is given in the table below.

$$\Sigma_j = \frac{N}{T}\sum_{t=1}^{T}\frac{x_t {x_t}^{\mathsf{T}}}{{x_t}^{\mathsf{T}}(\Sigma_{j-1})^{-1}x_t} \tag{46}$$

Table 5. Estimated Covariance Matrix of Publicly Listed Stocks in United States

Ticker	A	AA	⋯	ZYXI
A	0.00028431	0.00020095	⋯	0.00010035
AA	0.00020095	0.00107207	⋯	0.00028756
⋮	⋮	⋮	⋱	⋮
ZYXI	0.00010035	0.00028756	⋯	0.00156091

One may observe that the objective function of (34) is nonlinear and it could be treated as a quadratic programming model. However, some decision variables are binary and it would be hard to deduce the optimal solution. Accordingly, we provide only one algorithm for solving the problem.

The intention of the branch-and-bound method is to tactically search for the optimal solution of an integer programming problem by examining the upper bound or lower bound of the objective function (Land & Doig, 1960). It decomposes a discrete optimization problem into a list of subproblems which are expected to be tackled effortlessly via any well-established algorithm.

At first, we deal with the same problem without the integrality restriction. If its solution complies with the discrete requirement, the search stops as it already obtains the optimal solution. Otherwise, select a variable for branching in which the lower bound of this variable is the value of the relaxed solution after round-up and vice versa. This step divides the feasible region into two parts and each constitutes a unique subproblem (one for the lower bound constraint and another for the upper bound constraint). While branching produces a searching direction, the bounding step eliminates some impossible scenarios. By solving the linear programming relaxation together with the new realizable domain, the upper limit or lower limit at each node is determined by the new relaxed solution. At last, pick the subproblem that has a larger bound and replicate the branching and bounding procedure until an integer solution is reached. Because the main attention of this project is the integer form of the k-means clustering problem, more branching rules could be found in Billionnet et al. (2014) or Buchheim et al. (2010). In addition, *Gurobi* is the desired solver for optimizing the problem in stage two with the following parameters being used.

Table 6. List of Parameters Applied in Portfolio Optimization

(1) $L = 0.000005$	(4) $N_{C_k} = 10$
(2) $c = 0.015$	(5) $\beta = 0.0012$
(3) $t = 0.0008\%$	

Table 7. Price of Publicly Listed Stocks in United States on 02-01-2020

Ticker	Price
A	83.502022
AA	20.890644
⋮	⋮
ZYXI	7.121037

Chapter 5
Algorithms Analysis

This section describes the outcomes of the k-means clustering problem in stage one and analyzes the strengths and weaknesses of each algorithm which could be wound up from various perspectives.

First, the table below wraps up the number of stocks in each cluster. It is clear that both the Lloyd's algorithm and the trace maximization algorithm had accomplished the same result. Each cluster is comprised of identical stocks. Conversely, the clusters generated by the Frank-Wolfe algorithm deviates from the other two a lot.

Table 8. Number of Stocks in each Clusters

Clusters	Lloyd's Algorithm	Frank-Wolfe Algorithm	Trace Maximization
0	178	45	178
1	12	7	12
2	214	437	214
3	1	1	1
4	743	646	743
5	3	15	3
6	1	1	1

Second, all algorithms should be compared in line with their accuracy. For this reason, they had been run for multiple times. The minimum optimal objective function value captured by the Lloyd's algorithm is exactly replicated by the trace maximization algorithm as the previous table suggests. On the other hand, the Frank-Wolfe algorithm does not realize an acceptable minimum optimal objective function value because it is much higher than others. Despite that, it does obtain the lowest maximum optimal objective function value among all three algorithms. However, this is not a crucial element in deciding which algorithm is the best when we are minimizing the objective function instead of maximizing it.

Table 9. Statistics Summary of the Optimal Objective Function Value

Statistics	Lloyd's Algorithm	Frank-Wolfe Algorithm	Trace Maximization
Minimum	1.767911294	11.4017485	1.767911294
Maximum	25.69973077	21.2248505	27.77615746
Average	13.95167173	18.1871119	6.685401817
Deviation (σ)	10.25115024	1.27262664	4.789702741

All algorithms had been run for 500 times and the initial clustering centroids were rechosen randomly for each time

Values are highlighted in green to indicate the best clustering result. As discussed earlier, we may ignore the maximum optimal objective function value. Thus, the trace maximization algorithm is believed to outperform its opponents in view of having a lower average optimal objective function value as well as a smaller standard deviation of the optimal objective function value. To be more precise, the trace maximization algorithm could find the best solution more frequently due to its little variation from the mean optimal objective function value. Therefore, it would need to be repeated fewer number of times before the best clustering is reached. Up to this point, the conclusion raised by Nie et al. (2023) is confirmed where the trace maximization algorithm delivers a better clustering than the Lloyd's algorithm does.

Subsequent to the comparison of the clustering result, we turn our focus to the computational time of each algorithm for illustrating why the Frank-Wolfe algorithm is usually not applicable. Apart from its poor performance, its long computational time is the largest issue which is due to the fact that it is constructed by a double loop structure. It alternately refreshes two matrices (first loop) by recurrently computing the gradient matrix to update each matrix (second loop). Although the trace maximization algorithm is also made up of a double loop configuration, its short computational time counts on its fast convergence rate. On the contrary, the Lloyd's algorithm is a single loop algorithm and thereby could converge with a fast speed.

Table 10. Average Computational Time of each Algorithm

Statistics	Lloyd's Algorithm	Frank-Wolfe Algorithm	Trace Maximization
Time (s)	0.004037932	34.4242515	0.418781857

Table 11. Complexity of each Algorithm

Statistics	Lloyd’s Algorithm	Frank-Wolfe Algorithm	Trace Maximization
Complexity	$\mathcal{O}(N^{MK+1})$	$\mathcal{O}(1/t)$	$\mathcal{O}(NMK)$

N is the number of data point, M is the dimension of data, K is the number of clusters, and t is the number of iteration

The complexity of the Frank-Wolfe algorithm refers to the convergence rate for updating a single matrix (see figure 4)

An algorithm is judged to be an outstanding algorithm if it could tackle a problem within a very short period of time as well as having a notably high accuracy. Manifestly, the Frank-Wolfe algorithm is not an algorithm like that according to its performance. With the goal of boosting the precision of the model, many computer programs in this day and age would execute the algorithm numerous times. Considering the average duration for completing the Frank-Wolfe algorithm once, it would require roughly an hour to repeat a hundred of time which is exceedingly not tolerable.

In terms of empty cluster concern, the Lloyd’s algorithm and the trace maximization algorithm seem to have a superior prevention method. During their assignment process, we could divide the largest cluster into two smaller ones for avoiding the occurrence of an empty cluster (Nie et al., 2023). Moreover, some may reuse the previous indicator matrix and randomly pick one point to remain staying in the blank cluster in the current iteration. However, the Frank-Wolfe algorithm always generates a non-binary indicator matrix. As a result, it actually gives a probability distribution while a cluster is assigned to each data point based on the maximum likelihood. Therefore, it would be complex to introduce a method to keep away from the empty cluster issue.

The disappointing performance of the Frank-Wolfe algorithm had made it difficult to use in many instances. Yet, the small standard deviation of the optimal objective function value provides a hint for us to understand when this algorithm would be applicable. Notice that it is primarily used to solve the archetypal analysis problem. In other words, it estimates the convex hull of a series of data. Hence, the k-means clustering problem is analogous to the archetypal analysis when the center could form a convex hull. For example, a convex polygon is said to be a convex hull in a two-dimensional space. Bear in mind that financial variables could seldom be separated into different clusters with the centroid of each cluster would form a convex hull, the Frank-Wolfe algorithm is not practical in stock segmentation. Nonetheless, its accuracy should be enlarged substantially when the center of each cluster shapes a convex hull.

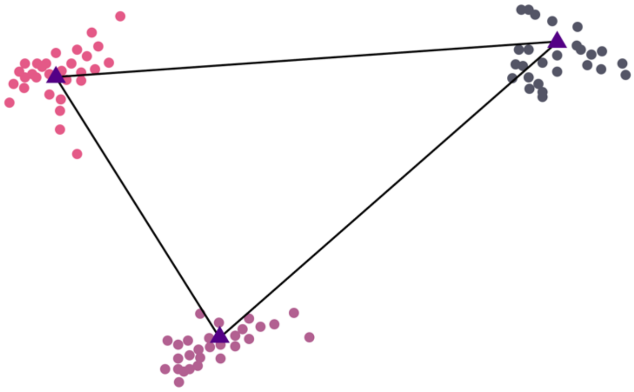

Figure 8. Example of Convex Hull in the Two-dimensional Space

Chapter 6
Portfolio Analysis

The following paragraphs outline the details of the portfolios we had obtained from the optimization model in stage two. First, the optimal weight of each stock is documented in the table below. Because the Lloyd's algorithm produced the same clustering result as the trace maximization algorithm did, the portfolios created by both algorithms is identical to each other. Therefore, the conclusion with respect to the Lloyd's algorithm portfolio is also applicable in the trace maximization algorithm portfolio in the later analysis Besides, we may expect that the Frank-Wolfe algorithm would construct a partially different portfolio owing to the mismatch in the building blocks in each cluster.

Table 12. Asset Allocation in each Portfolio

Stocks	Lloyd's Algorithm	Frank-Wolfe Algorithm	Trace Maximization
ALTG-PA	0.04537959	0.04897986	0.04537959
ANF	0.00497333	0	0.00497333
BVN	0.00142030	0	0.00142030
CHT	0.02267924	0.00880946	0.02267924
CMRE-PB	0.04239139	0.03897900	0.04239139
CMRE-PE	0	0.03947824	0
COKE	0	0.01383651	0
COR	0.07864473	0.07360765	0.07864473
CSPI	0	0.00029696	0
FATBB	0.01097836	0.01237984	0.01097836
FERG	0.01092284	0.01571881	0.01092284
FLXS	0.00791385	0.00864075	0.00791385
FORTY	0.00955732	0	0.00955732
GLP-PB	0.21475466	0.20373702	0.21475466

GYRE	0.00501048	0.00555238	0.00501048
IMPPP	0.02347944	0.02368095	0.02347944
KRT	0.00821986	0	0.00821986
LANDM	0.03255538	0.03309331	0.03255538
LLY	0.02509039	0.02823988	0.02509039
MDV	0.00885037	0.0088779	0.00885037
MUSA	0.00554338	0	0.00554338
POWWP	0.06807006	0.07316357	0.06807006
RDY	0.02877230	0.02754569	0.02877230
RSG	0.03278332	0.02820329	0.03278332
SB-PD	0.04391341	0.03465260	0.04391341
SNT	0	0.00007912	0
TNP	0.00869838	0	0.00869838
TNP-PE	0.12880870	0.13059359	0.12880870
VATE	0.03646291	0.03764898	0.03646291
VEON	0	0.00067480	0
VST	0	0.00837967	0
WCC-PA	0.08117314	0.08084774	0.08117314
WLYB	0.01295287	0.01430244	0.01295287

Stocks with the highest optimal weight in that portfolio are highlighted in red

After tabulating the asset allocation of each portfolio, we point out an essential discovery. The portfolio constructed based on the clustering result of the Lloyd's algorithm is similar to that according to the clustering outcome of the Frank-Wolfe algorithm. To be specific, majority of the stocks found in the Lloyd's algorithm portfolio are also the constituents of the Frank-Wolfe algorithm portfolio. In addition, their optimal weights are nearly equivalent with a maximum fluctuation of two percent except for one stock. The reason behind this phenomenon could be deduced from the following conditional probability. Consider the likelihood of a stock that would have a nonzero weight in both portfolios and given that it had already been assigned to the same cluster by both algorithms. Unless this possibility is low, we would not have obtained the above asset allocation. In other words, the number of stocks being selected to be a part of the portfolios is very likely to be assigned to the same group by both algorithms.

Beyond the aforementioned observation, a vital purpose of using the mixed-integer programming approach is to impede the development of an extreme portfolio. Our proposed method was undoubtedly a great success where the maximum asset allocation in all portfolios is no more than a quarter of the initial total wealth. Thus, such type of portfolios did not form in our method. Furthermore, the risk of the Lloyd's algorithm portfolio and the Frank-Wolfe algorithm portfolio are slightly lower than that of the original Markowitz's portfolio. However, we had achieved a simpler portfolio because of the idea of only selecting a part of the stocks in the same clusters.

Table 13. Statistics Summary of each Portfolio

Statistics	Lloyd's Algorithm	Frank-Wolfe Algorithm	Trace Maximization
Return	0.001199	0.001199	0.001199
Illiquidity	0.000003	0.000003	0.000003
Risk	0.000011	0.000011	0.000011

The risk, illiquidity, and return of the Markowitz's portfolio are 0.001200, 0.000005, and 0.000011 respectively

The Markowitz's portfolio is a portfolio without the clustering constraint. Other restrictions such as illiquidity remain unchanged

Portfolios were also backtested to ensure the effectiveness of our proposed strategy and their values are plotted on the following graph to track their performance. First, the actual returns of all portfolios are much greater than their corresponding expected returns. Second, the return of the Lloyd's algorithm portfolio is similar to that of the Frank-Wolfe algorithm portfolio given their optimal asset allocations are almost the same. Third, none of them would have a negative portfolio value after the backtesting. What is more, the average annual returns of our portfolios are very close to some indices such as the Standard and Poor's one but with a smaller number of stocks. Therefore, our design is believed to be powerful.

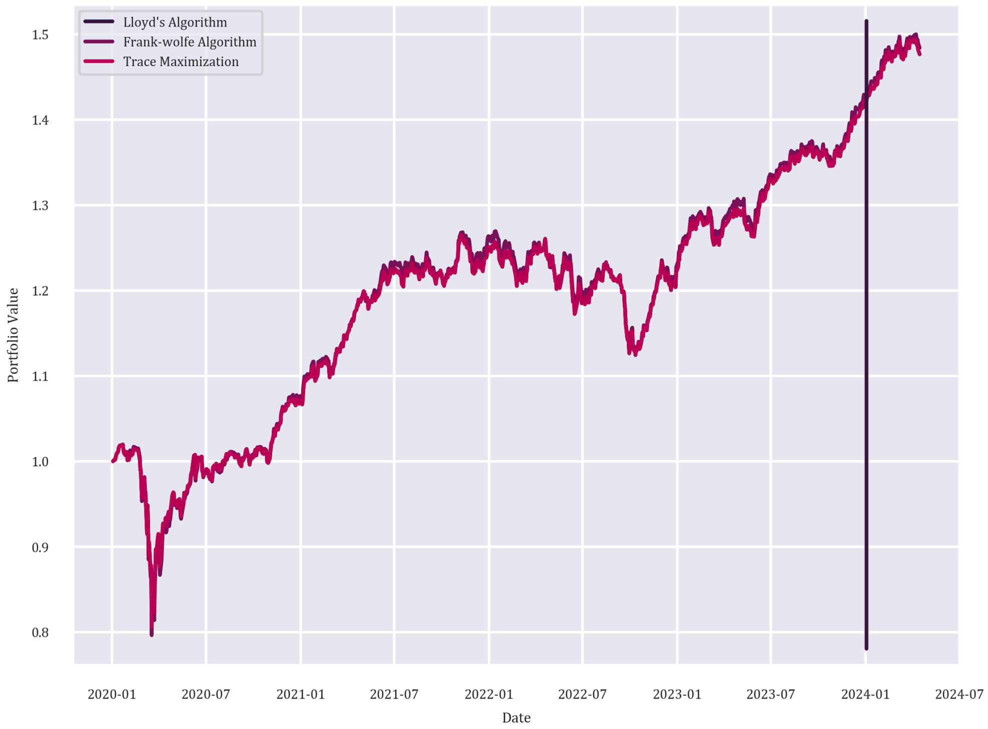

Figure 9. Portfolio Values

Investors carry out Monte Carlo simulation to estimate the expected value at risk for understanding how likely their portfolio values would fall under a certain number. The last analysis in this project also follows this procedure in which we assume the movement of stock price is a Geometric Brownian Motion (Reddy & Clinton). In other words, they follow a lognormal distribution as shown in (47)

$$S_T = S_0 e^{\left(\mu - \frac{1}{2}\sigma^2\right)T + \sigma\Phi\sqrt{T}} \tag{46}$$

where Φ is a standard normal variable. Unquestionably, the simulation result below advises that all portfolios would have a stable growth during the investment period. Unlike the ritualistic Markowitz's portfolio, unanticipated turnover would not be too large. The maximum expected loss would be no more than one-fifth of the total wealth.

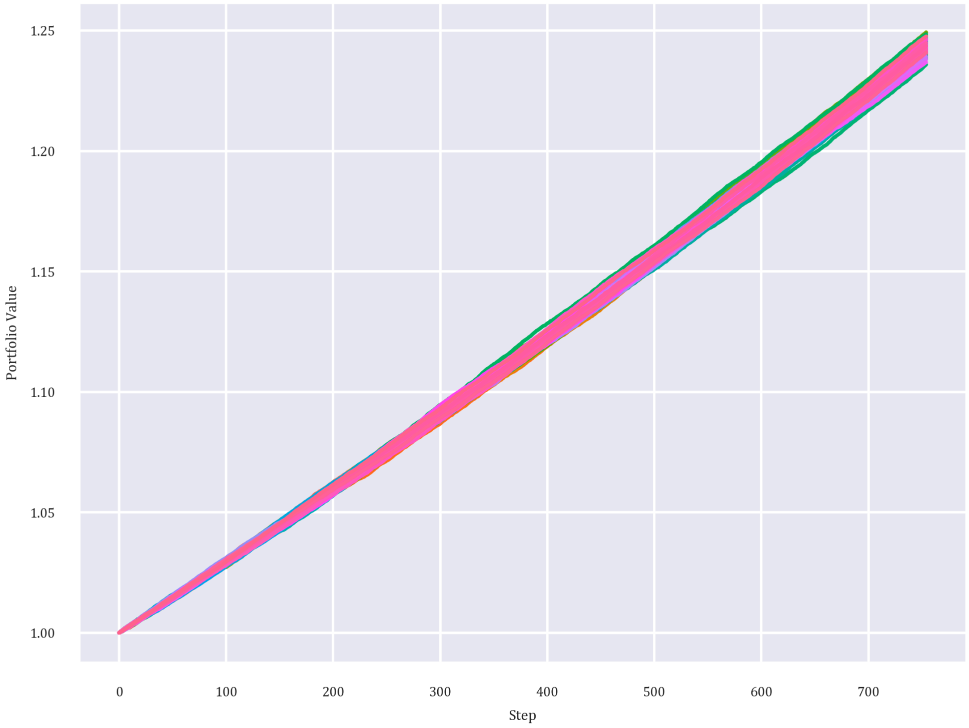

Figure 10. Monte Carlo Simulation for the Lloyd's Algorithm Portfolio

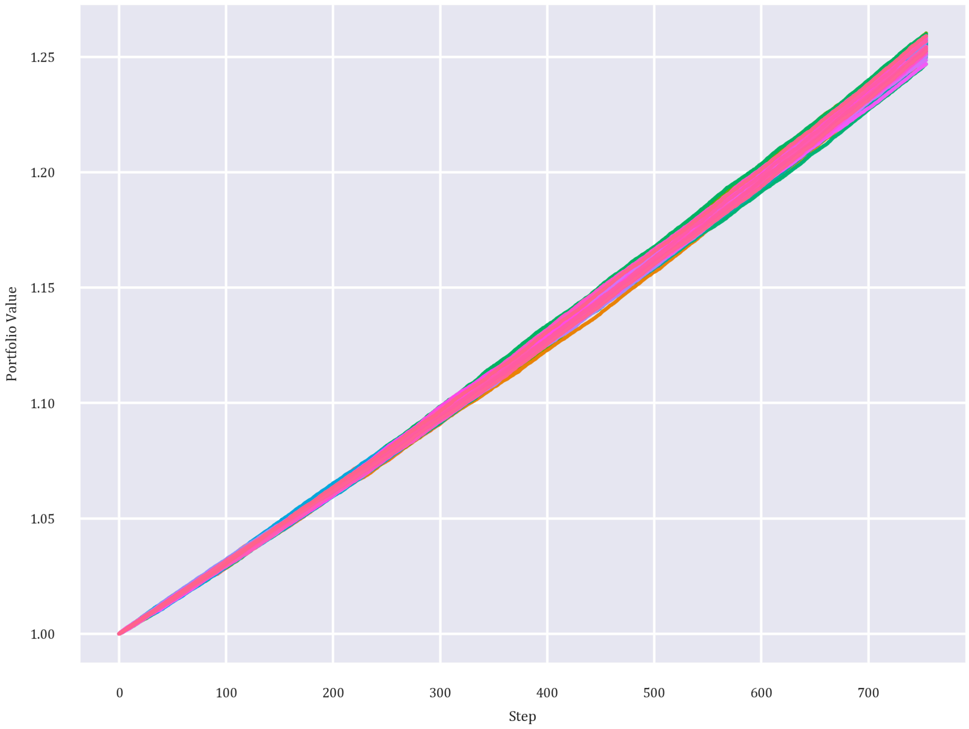

Figure 11. Monte Carlo Simulation for the Frank-Wolfe Algorithm

Chapter 7
Conclusion

This project had compared three different algorithms in solving the integer programming form of the k-mean clustering problem. On top of this, the trace maximization algorithm is the most desirable one. For those who would process a vast volume of data, especially financial institutions, are recommended to use the trace maximization algorithm instead of the Lloyd's algorithm due to its strong ability (fast convergence rate and high accuracy) to handle a large dataset. In contrast, other analysts such as individual investors could use either the Lloyd's algorithm or the trace maximization algorithm as there is no material distinction between them when the dataset is small.

In general, we also demonstrated a resilient approach for portfolio construction with the help of the k-means clustering technique. This helps to identify a set of uncorrelated securities for constructing a potential diversified portfolio. Realistic constraints like liquidity and transaction costs are taken into account in portfolio construction. In short, k-means clustering could be exploited as a stock recommendation systems. Using the two-stage mixed integer programming structure promoted in this work, the mission of diversification or risk minimization had achieved. Meanwhile, we are confident that our approach is simple enough and able to produce a profitable portfolio while clustering could provide insight into finding uncorrelated securities that may be missed or ignored in human analysis which support decision making.

Nevertheless, we would like to list out some limitations in our work. First, the dataset being considered in this paper only included publicly listed stocks in United States. Other asset classes such as private equities and bonds should not be omitted as a means to mitigate the risk as much as possible. Second, we ran all algorithms by randomly choosing the centroids of each clusters. However, the method to initialize these centers may marginally affect the analysis. So these algorithms should also be tested by altering the initial centroids selection method. For instance, k-means++ initialization.

Future works could further investigate the same portfolio construction approach but expanding the dataset by including other asset classes such as commodities and derivatives. Researchers could also adopt other advanced techniques in stage one for better clustering. For example, the deep learning network in an artificial intelligence model.

References

Aguadé, J., & Kane, R. M. (1987). *Algebraic topology, Barcelona 1986: proceedings of a symposium held in Barcelona, April 2-8, 1986*. Springer-Verlag. https://doi.org/10.1007/BFb0082996

Amihud, Y. (2002). Illiquidity and stock returns: cross-section and time-series effects. *Journal of Financial Markets (Amsterdam, Netherlands)*, *5*(1), 31-56. https://doi.org/10.1016/S1386-4181(01)00024-6

Baksalary, O. M., & Trenkler, G. (2014). On subspace distances determined by the Frobenius norm. *Linear Algebra and Its Applications*, *448*, 245-263. https://doi.org/10.1016/j.laa.2014.01.017

Bertsimas, D., Darnell, C., & Soucy, R. (1999). Portfolio Construction Through Mixed-Integer Programing at Grantham, Mayo, Van Otterloo and Company. *Interfaces*, *29*(1), 49-66. https://doi.org/10.1287/inte.29.1.49

Billionnet, A., Elloumi, S., & Lambert, A. (2014). A Branch and Bound algorithm for general mixed-integer quadratic programs based on quadratic convex relaxation. *Journal of Combinatorial Optimization*, *28*(2), 376-399. https://doi.org/10.1007/s10878-012-9560-1

Bomze, I. M., Rinaldi, F., & Zeffiro, D. (2020). Active Set Complexity of the Away-Step Frank-Wolfe Algorithm. *SIAM Journal on Optimization*, *30*(3), 2470-2500. https://doi.org/10.1137/19M1309419

Buchheim, C., Caprara, A., & Lodi, A. (2010). An Effective Branch-and-Bound Algorithm for Convex Quadratic Integer Programming. *Integer Programming and Combinatorial Optimization*. *6080*, 285-298. Springer. https://doi.org/10.1007/978-3-642-13036-6_22

Christian, B. (2016). K-means Clustering via the Frank-Wolfe Algorithm. *Central Europe Workshop Proceedings*, *1670*, 311-322. https://ceur-ws.org/Vol-1670/paper-65.pdf

Christian, B. (2020). NumPy / SciPy Recipes for Data Science: Archetypal Analysis via Frank-Wolfe Optimization. *ResearchGate*. https://www.researchgate.net/publication/344671912_NumPy_SciPy_Recipes_for_Data_Science_Archetypal_Analysis_via_Frank-Wolfe_Optimization

Eugster, M. J. A., & Leisch, F. (2009). From Spider-Man to Hero - Archetypal Analysis in R. *Journal of Statistical Software, 30*(8). https://doi.org/10.18637/jss.v030.i08

Fang, Z., & Chiao, C. (2021). Research on prediction and recommendation of financial stocks based on K-means clustering algorithm optimization. *Journal of Computational Methods in Sciences and Engineering*, *21*(5), 1081-1089. https://doi.org/10.3233/JCM-204716

Fulga, C. (2015). Integrated bi-criteria decision support system for portfolio selection. *Journal of Decision Systems*, *24*(2), 159-177. https://doi.org/10.1080/12460125.2015.1046684

Gray, R. M. (2011). *Entropy and Information Theory*. SpringerLink. https://doi.org/10.1007/978-1-4419-7970-4

Huber, P. J., & Ronchetti, E. (2009). *Robust statistics* (2nd ed.). Wiley. https://onlinelibrary-wiley-com.lib.ezproxy.hkust.edu.hk/doi/book/10.1002/9780470434697

Ji, Y., Wang, T., Goh, M., Zhou, Y., & Zou, B. (2014). The worst-case discounted regret portfolio optimization problem. *Applied Mathematics and Computation*, *239*, 310-319. https://doi.org/10.1016/j.amc.2014.04.072

Land, A. H., & Doig, A. G. (1960). An Automatic Method of Solving Discrete Programming Problems. *Econometrica*, *28*(3), 497-520. https://doi.org/10.2307/1910129

Markowitz, H. (1952). Portfolio Selection. *The Journal of Finance*, *7*(1), 77-91. https://doi.org/10.2307/2975974

Merle, O. D., Hansen, P., Jaumard, B., & Mladenovic, N. (1999). An interior point algorithm for minimum sum-of-squares clustering. *Society for Industrial and Applied Mathematics Journal on Scientific Computing*, *21*(4), 1485-1505. https://doi.org/10.1137/S1064827597328327.

Michaud, R. O. (1998). *Efficient Asset Management: A Practical Guide to Stock Portfolio Optimization and Asset Allocation*. Oxford University Press.

Nie, F., Li, Z., Wang, R., & Li, X. (2023). An Effective and Efficient Algorithm for K-Means Clustering With New Formulation. *IEEE Transactions on Knowledge and Data Engineering*, *35*(4), 3433–3443. https://doi.org/10.1109/TKDE.2022.3155450

Nie, F., Wang, X., & Huang, H. (2017). Multiclass Capped ℓ_p-Norm SVM for Robust Classifications. *Proceedings of the AAAI Conference on Artificial Intelligence*, *31*(1). https://doi.org/10.1609/aaai.v31i1.10948

Rebollo-Monedero, D., Forne, J., Pallares, E., & Parra-Arnau, J. (2013). A modification of the Lloyd algorithm for k-anonymous quantization. *Information Sciences*, *222*, 185-202. https://doi.org/10.1016/j.ins.2012.08.022

Reddy, K., & Clinton, V. (2016). Simulating Stock Prices Using Geometric Brownian Motion: Evidence from Australian Companies. *Australasian Accounting, Business & Finance Journal*, *10*(3), 23-47. https://doi.org/10.14453/aabfj.v10i3.3

Ying Sun, Babu, P., & Palomar, D. P. (2014). Regularized Tyler's Scatter Estimator: Existence, Uniqueness, and Algorithms. *IEEE Transactions on Signal Processing*, *62*(19), 5143-5156. https://doi.org/10.1109/TSP.2014.2348944

Ying Sun, Babu, P., & Palomar, D. P. (2015). Regularized Robust Estimation of Mean and Covariance Matrix Under Heavy-Tailed Distributions. *IEEE Transactions on Signal Processing*, *63*(12), 3096-3109. https://doi.org/10.1109/TSP.2015.2417513

Zhu, Z., & Liu, N. (2021). Early Warning of Financial Risk Based on K-Means Clustering Algorithm. *Complexity*, *2021*, 1-12. https://doi.org/10.1155/2021/5571683

Appendix

Appendix A
Python Code for Lloyd’s Algorithm

Data

```
import pandas, numpy, time

data = pandas.read_csv("Data.csv", index_col = "Ticker")

data = data.dropna()

print(data)
```

Normalization

```
sklearn.preprocessing import MinMaxScaler

train = MinMaxScaler().fit_transform(data)

train = pandas.DataFrame(train, index = data.index)

print(train)
```

K-means Clustering

```
from sklearn.cluster import KMeans

def kmeans(x_train, k):

    model = KMeans(n_clusters = k, n_init = 1, init = "random”).fit(x_train)

    return [model.inertia_, model.labels_]
```

Result

```
length, result = time.time(), {}

for i in range(500):

    result[i] = kmeans(train, 7)

length = time.time() - length
```

```
index = numpy.argmin([result[i][0] for i in result])

colum = ["Clusters"]

frame = pandas.DataFrame(result[index][1], columns = colum, index = train.index)

print(frame)
```

Appendix B

Python Code for Frank-Wolfe Algorithm

Data

```python
import pandas, numpy, time

data = pandas.read_csv("Data.csv", index_col = "Ticker")

data = data.dropna()

print(data)
```

Normalization

```python
sklearn.preprocessing import MinMaxScaler

train = MinMaxScaler().fit_transform(data)

train = pandas.DataFrame(train, index = data.index)

print(train)
```

Frank-Wolfe Algorithm

```python
def matrix_y(mat_xtx, mat_xxz, mat_zzt, mat_y):

    index = numpy.arange(mat_y.shape[1])

    for i in range(100):

        gamma = 2 * ((i + 2) ** -1)

        mat_g = 2 * (numpy.dot(numpy.dot(mat_xtx, mat_y), mat_zzt) - mat_xxz) -
            numpy.log(mat_y, where = mat_y > 0)

        argmn = numpy.argmin(mat_g, axis = 0); mat_y = mat_y * (1 - gamma)

        mat_y[argmn,index] = mat_y[argmn,index] + gamma

    return mat_y
```

```
def matrix_z(mat_mtm, mat_mtx, mat_z):

    index = numpy.arange(mat_z.shape[1])

    for i in range(100):

        gamma = 2 * ((i + 2) ** -1)

        mat_g = numpy.dot(numpy.dot(mat_mtm, mat_z) - mat_mtx

        argmn = numpy.argmin(mat_g, axis = 0); mat_z = mat_z * (1 - gamma)

        mat_z[argmn,index] = mat_z[argmn,index] + gamma

    return mat_z
```

Archetypal Analysis for K-means Clustering

```
def archetypal_analysis(mat_x, k):

    mat_m = mat_x[:,numpy.random.choice(mat_x.shape[1], k, replace = False)]

    # m, n = max_x.shape

    mat_y = numpy.zeros((mat_x.shape[1], k)); mat_y[0] = 1

    mat_z = numpy.zeros((k, mat_x.shape[1])); mat_z[0] = 1

    error = numpy.inf

    m_xtx = numpy.dot(numpy.transpose(mat_x), mat_x)

    for i in range(300):

        mat_mtm, mat_mtx = numpy.dot(numpy.transpose(mat_m), mat_m),
            numpy.dot(numpy.transpose(mat_m), mat_x)

        mat_z = matrix_z(mat_mtm, mat_mtx, mat_z)

        mat_zzt, mat_xxz = numpy.dot(mat_z, numpy.transpose(mat_z)),
            numpy.dot(m_xtx, numpy.transpose(mat_z))

        mat_y = matrix_y(m_xtx, mat_xxz, mat_y)

        mat_m = numpy.dot(mat_x, mat_y)

        if numpy.abs(error - numpy.linalg.norm(mat_x - numpy.dot(mat_m, mat_z)))
            < 10 ** -5:

            break

        error = numpy.linalg.norm(mat_x - numpy.dot(mat_m, mat_z))

    mat_mtm, mat_mtx = numpy.dot(numpy.transpose(mat_m), mat_m),
        numpy.dot(numpy.transpose(mat_m), mat_x)

    mat_z = matrix_z(mat_mtm, mat_mtx, mat_z)

    return mat_m, mat_y, mat_z
```

Result

```
length, result = time.time(), {}

for i in range(500):

    temps = archetypal_analysis(numpy.array(numpy.transpose(train)), 7)[2]

    index = numpy.argmax(temps, axis = 0)

    dataframes = pandas.concat([train, pandas.DataFrame(index, columns =
        ["Clusters"], index = train.index)], axis = 1)

    dataframes = dataframes.groupby("Clusters").var(ddof = 0)

    count = numpy.unique(index, return_counts = True)[1]

    result[i] = [numpy.sum(count @ dataframes), index]

length = time.time() - length
```

```
index = numpy.argmin([result[i][0] for i in result])

colum = ["Clusters"]

frame = pandas.DataFrame(result[index][1], columns = colum, index = train.index)

print(frame)
```

Appendix C

Python Code for Trace Maximization Algorithm

Data

```
import pandas, numpy, time

data = pandas.read_csv("Data.csv", index_col = "Ticker")

data = data.dropna()

print(data)
```

Normalization

```
sklearn.preprocessing import MinMaxScaler

train = MinMaxScaler().fit_transform(data)

train = pandas.DataFrame(train, index = data.index)

print(train)
```

Trace Maximization Algorithm

```
def vector_y(mat_x, mat_z):

    vec_z = numpy.transpose(mat_z)

    vec_v = numpy.array([])

    for i in vec_v:

        value = numpy.sqrt(i @ numpy.transpose(mat_x) @ mat_x @ i)

        value = value * ((i @ i) ** -1)

        vec_v = numpy.append(vec_v, value)

    return vec_v
```

```
def matrix_q(mat_x, mat_z):

    vec_z = numpy.transpose(mat_z)

    mat_q = numpy.array([])

    for i in vec_z:

        value = numpy.sqrt(i @ numpy.transpose(mat_x) @ mat_x @ i) ** -1

        value = (numpy.transpose(mat_x) @ mat_x @ i)  * value

        mat_q = numpy.append(mat_q, value)

    return numpy.transpose(numpy.reshape(mat_q, (len(vec_z), -1)))
```

```
def matrix_d(mat_x, mat_q, vec_v):

    mat_v = numpy.zeros((len(vec_v), len(vec_v)))

    for i in range(len(vec_v)):

        mat_v[i][i] = vec_v[i]

    mat_a = 2 * mat_q @ mat_v

    mat_o = mat_v ** 2

    mat_o = numpy.ones((len(numpy.transpose(mat_x)), len(mat_v))) @ mat_o

    return mat_a - mat_o
```

```
def update_z(mat_x, index, k):

    diffs = list(set(numpy.arange(k)) - set(numpy.unique(index)))

    for i in range(len(vec_v)):

        mat_v[i][i] = vec_v[i]

    mat_a = 2 * mat_q @ mat_v

    mat_o = mat_v ** 2

    mat_o = numpy.ones((len(numpy.transpose(mat_x)), len(mat_v))) @ mat_o

    return mat_a - mat_o
```

```
def trace_maximization(mat_x, k):

    index = numpy.random.choice(k, len(numpy.transpose(mat_x)))

    mat_z = numpy.zeros((len(numpy.transpose(mat_x)), k))

    for i in range(len(index)):

        mat_z[i][index[i]] = 1

    vec_v = vector_v(mat_x, mat_z); mat_q = matrix_q(mat_x, mat_z)

    m_a_b = matrix_d(mat_x, mat_q, vec_v)

    for i in range(300):

        upd_z = update_z(mat_x, numpy.argmax(m_a_b, axis = 1), k)

        for j in range(100):

            mat_q = matrix_q(mat_x, upd_z); m_a_b = matrix_d(mat_x, mat_q, vec_v)

            new_z = update_z(mat_x, numpy.argmax(m_a_b, axis = 1), k)

            if numpy.linalg.norm(new_z - upd_z) < 0.00001:

                upd_z = new_z

                break

            upd_z = new_z

        if numpy.linalg.norm(upd_z - mat_z) < 0.00001:

            return upd_z

        mat_z = upd_z

        vec_v = vector_v(mat_x, mat_z)

    return mat_z
```

Result

```
length, result = time.time(), {}

for i in range(500):

    temps = trace_maximization(numpy.array(numpy.transpose(train)), 7)

    index = numpy.argmax(temps, axis = 1)

    dataframes = pandas.concat([train, pandas.DataFrame(index, columns = ["Clusters"],
        index = train.index)], axis = 1)

    dataframes = dataframes.groupby("Clusters").var(ddof = 0)

    count = numpy.unique(index, return_counts = True)[1]

    result[i] = [numpy.sum(count @ dataframes), index]

    length = time.time() - length
```

```
index = numpy.argmin([result[i][0] for i in result])

colum = ["Clusters"]

frame = pandas.DataFrame(result[index][1], columns = colum, index = train.index)

print(frame)
```

Appendix D

Python Code for Portfolio Optimization

Data

```
import pandas, numpy, gurobipy, datetime

price = pandas.read_csv("Price.csv", index_col = "Date")

index = numpy.array([])

for i in price.index:

    index = numpy.append(index, datetime.datetime.strptime(i, "%Y-%m-%d %H:%M:%S"))

price = price.set_index(index)

print(price)
```

Robust Estimators to Heavy Tails

```
returns = price.interpolate(method="time").apply(numpy.log).apply(numpy.diff).dropna()

from scipy.stats import trim_mean

mat_m = numpy.array([trim_mean(returns[i], 0.05) for i in returns.columns])

print(pandas.DataFrame(mat_m, price.columns))
```

```
from sklearn.covariance import EmpiricalCovariance

variance = pandas.DataFrame(EmpiricalCovariance().fit(returns).covariance_)

variance.index = price.columns; variance.columns = price.columns

print(variance)
```

Data Cleaning

```
mat_k, mat_a, mat_b = pandas.read_csv("Clustering.csv", index_col = "Ticker"), {}, {}

for i in mat_k.columns:

    p = numpy.transpose(price)["2020-01-02 00:00:00"]

    a = numpy.array([i ** -1 if i != 0 else 0 for i in list(p.fillna(value = 0))])

    m = mat_m - 8.0e-6 - 1.5e-2 * a

    l = numpy.array(pandas.read_csv("Liquidity.csv", index_col = "Date").mean())

    mat_a[i] = {0: m, 1: l}; mat_b[i] = [1.2e-3, 5.0e-6]
```

```
mat_k, mat_c, mat_d = pandas.read_csv("Clustering.csv", index_col = "Ticker"), {}, {}

for i in mat_k.columns:

    temp_1 = numpy.zeros((numpy.max(mat_k[i]) + 1, len(mat_m)))

    temp_3 = numpy.ones(numpy.max(mat_k[i]) + 1) * 1.0e0

    temp_2 = numpy.zeros((numpy.max(mat_k[i]) + 1, len(mat_m))); temp_4 =

    temp_4 = numpy.ones(numpy.max(mat_k[i]) + 1) * 1.0e1

    for j in range(len(mat_k[i])):

        temp_1[list(mat_k[i])[j]][j] += 1

        temp_2[list(mat_k[i])[j]][j] += 1

    mat_c[i] = {0: temp_1, 1: temp_2}

    mat_d[i] = {0: temp_3, 1: temp_4}
```

Portfolio Optimization

```
from gurobipy import GRB

def portfolio_optimization(mat_a, mat_b, mat_c, mat_d, mat_p, mat_s):

    result, mat_ones = {}, numpy.ones(len(mat_a[0]))

    portfolio = gurobipy.Model()

    variables = {"w": portfolio.addMVar(len(mat_a[0]), lb = 0), "b":
                      portfolio.addMVar(len(mat_a[0]), vtype = GRB.BINARY)}

    portfolio.addConstr(mat_a[0] @ variables["w"] >= mat_b[0])

    portfolio.addConstr(mat_a[1] @ variables["w"] <= mat_b[1])

    portfolio.addConstr(mat_c[0] @ variables["b"] >= mat_d[0])

    portfolio.addConstr(mat_c[1] @ variables["b"] <= mat_d[1])

    portfolio.addConstr(mat_ones @ variables["w"] == 1);

    portfolio.addConstr(variables["w"] <= variables["b"])

    for k in range(len(mat_p)):

        if k == 0: portfolio.addConstr(variables["b"][k] == 0)

    portfolio.setObjective(variables["w"] @ numpy.array(mat_s) @ variables["w GRB.MINIMIZE)"]

    portfolio.update(); portfolio.optimize()

    if portfolio.status == 2:

        result["o"] = portfolio.getObjective().getValue()

        result["w"] = (variables["w"]).x

        return result

    return "Error"
```

Result

```
vec_pr = numpy.transpose(price)["2020-01-02 00:00:00"]

vec_pr = numpy.array([i ** -1 if i != 0 else 0 for i in list(vec_pr.fillna(value = 0))])

result = {}

for i in mat_k.drop(["Trace Maximization"], axis = 1).columns:

    result[i] = portfolio_optimization(mat_a[i], mat_b[i], mat_c[i], mat_d[i], vec_pr, variance)
```

```
dataframe, keys = {}, list(result.keys())

for i in range(len(keys)):

    r = mat_a[keys[i]][0] @ result[keys[i]]["w"]

    l = mat_a[keys[i]][1] @ result[keys[i]]["w"]

    dataframe[keys[i]] = {"Return": r, "Illiquidity": l, "Risk": result[keys[i]]["o"]}

print(pandas.DataFrame(dataframe))
```

```
weight = {i: result[i]["w"] for i in result}

weight = pandas.DataFrame(weight)

weight["Trace Maximization"] = weight["Lloyd's Algorithm"]

weight.to_csv("Portfolio.csv")
```

Appendix E

Python Code for Monte Carlo Simulation

Data

```
import pandas, numpy, datetime, matplotlib, seaborn

weight = pandas.read_csv(“Portfolio.csv”, index_col = “Ticker”)

print(weight)

weight = numpy.transpose(weight)
```

```
portfolio = pandas.DataFrame()

for i in weight.columns:

    if numpy.any(numpy.array(weight[i])):

        portfolio = pandas.concat([portfolio, pandas.DataFrame(weight[i]).T])
```

Data Cleaning

```
price = pandas.read_csv("Price.csv", index_col = "Date")

stock = numpy.array([])

for i in price.index:

    stock = numpy.append(stock, datetime.datetime.strptime(i, "%Y-%m-%d %H:%M:%S"))

    price = price.set_index(stock)

print(price)
```

Portfolio Value

```
returns = price.interpolate(method = "time").apply(numpy.log).apply(numpy.diff).fillna(value = 0)

por_val = {}

mat_m, variance = returns.mean(), returns.cov()

for i in numpy.transpose(weight):

    por_val[i] = numpy.transpose(weight)[i] @ numpy.transpose(returns)

por_val = pandas.DataFrame(por_val)

plot_va = {i: numpy.array([]) for i in por_val}

for i in por_val.columns:

    for j in range(len(por_val[i])):

        if j == 0: plot_va[i] = numpy.append(plot_va[i], 1)

        if j != 0: plot_va[i] = numpy.append(plot_va[i], numpy.exp(por_val[i][j])
                                    * plot_va[i][j-1])

ticks = list(price.index)[1:]

ax = seaborn.lineplot(pandas.DataFrame(plot_va, ticks), dashes = False)

ax.axvline(x = 19725, ymax = 0.975, ymin = 0.025)

ax.legend()

ax.set(xlabel = "Date", ylabel = "Portfolio Value")
```

Simulation

```
def geometric_browian(t, nsteps, npaths, price, mu, sigma):

    steps = t * (nsteps ** -1)

    paths = numpy.zeros((npaths, nsteps))

    paths[:,0] = price

    for i in range(nsteps - 1):

        paths[:,i+1] = paths[:,i] * numpy.exp((mu - 0.5 * (sigma ** 2)) * steps + sigma
                           * numpy.sqrt(steps) * numpy.random.randn(npaths))

   return paths
```

```
simulation = {}

for i in range(100):

    temper = {}; p = price.fillna(value = 0); m = pandas.DataFrame(mat_m)

    for j in weight.columns:

        arr = {"timet": 756, "nstep": 756, "npath": 1, "price": p[j]["2020-01-02"],
               "means": numpy.transpose(m)[j][0], "sigma": variance[j][j]}

        temper[j] = geometric_browian(arr["timet"], arr["nstep"], arr["npath"],
                                      arr["price"], arr["means"], arr["sigma"])[0]

   simulation[i] = pandas.DataFrame(temper).apply(numpy.log).apply(numpy.diff).fillna(value = 0)
```

```
result = {i: {} for i in numpy.transpose(weight)}

for i in numpy.transpose(weight):

    for j in simulation:

        temp = numpy.array(numpy.transpose(weight)[i]) @
               numpy.array(numpy.transpose(simulation[j]))

        arrs = numpy.array([])

        for k in range(len(temp)):

            if k == 0: arrs = numpy.append(arrs, 1)

            if k != 0: arrs = numpy.append(arrs, numpy.exp(temp[k]) * arrs[k-1])

        result[i][j] = arrs

print(result)
```

```
lloyd = result["Lloyd's Algorithm"]

ax = seaborn.lineplot(pandas.DataFrame(lloyd), dashes = False, legend = False)

ax.set(xlabel = "Step", ylabel = "Portfolio Value")
```

```
frank = result["Frank-wolfe Algorithm"]

ax = seaborn.lineplot(pandas.DataFrame(frank), dashes = False, legend = False)

ax.set(xlabel = "Step", ylabel = "Portfolio Value")
```