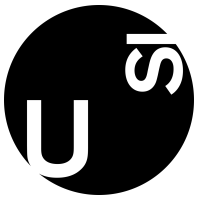

Università della Svizzera Italiana (USI)
Master of Science in Informatics

Università degli Studi di Milano-Bicocca (UniMiB)
Laurea Magistrale in Informatica (LM-18)

Enhancing Financial Client Segmentation Models through Time-Series Clustering

Author: Claudio Ricci
Student nr: 918956

Advisor: Prof. Olaf Schenk
Co-Advisor: Dr. Dimosthenis Pasadakis

Academic Year 2024/25

Abstract

Client segmentation is a fundamental task in financial services, enabling institutions to tailor products, enhance customer satisfaction and strengthen risk management. Traditional segmentation approaches often rely on static demographic or firmographic attributes, which fail to capture the behavioral diversity of clients. To address this limitation, this thesis advances customer segmentation methodologies by leveraging innovative techniques in time-series clustering.

Utilizing simulated customer and transaction data, the study develops an unsupervised, graph-based framework for client segmentation. Account behavior is modeled through time series derived from financial data, from which conditional dependencies are inferred using the *Sparse QUadratic approximation for Inverse Covariance* (SQUIC) and SQUIC-Fit algorithms. These dependencies are represented as graphs and multiple clustering methods are applied to uncover communities of behaviorally similar clients.

The evaluation shows that the proposed framework can generate client segments in settings where only aggregate balance data are available, simulating scenarios in which detailed transaction records cannot be accessed. The approach further demonstrates scalability across datasets of increasing size, indicating potential for application in realistic financial contexts.

Contents

Figures

Tables

Chapter 1

Introduction

Client segmentation is a central task in modern financial services and marketing; it refers to the process of dividing a heterogeneous client base into homogeneous subgroups in order to tailor strategies, services and products to the specific needs of each segment [1]. This practice is particularly significant in banking, where differentiated treatment of client groups enables institutions to improve profitability, enhance customer satisfaction, design customized financial products and strengthen monitoring and compliance strategies [2, 3]. By identifying structurally coherent segments, banks and other financial institutions can allocate resources more effectively, strengthen customer loyalty and improve risk management.

Traditional segmentation approaches often rely on static demographic or firmographic attributes such as income, firm size, or geographic region. However, these approaches fail to capture the dynamic and contextual nature of client behavior [1], business clients of comparable size may diverge significantly in their preferences for banking services, investment strategies or credit usage. Consequently, static segmentation criteria may obscure meaningful behavioral heterogeneity, limiting the strategic value of segmentation outcomes [4, 5].

Recent research emphasizes the importance of behavioral and dynamic segmentation approaches. Approaches based on clustering algorithms [6, 7] have proven effective in uncovering natural groupings in customer datasets, particularly when applied to behavioral indicators such as monetary transactions. By focusing on behavioral data, these methods adapt more readily to evolving client patterns and provide actionable insights for decision-making.

To address the challenge of customer segmentation in contexts with limited transactional visibility, this thesis proposes a novel unsupervised framework that employs sparse inverse covariance estimation and graph-based clustering techniques applied to time-series financial data. Rather than relying on explicit transaction records, the approach models behavioral similarity using only temporal account balance data, reflecting real-world scenarios in which transaction-level information may be inaccessible. A practical motivation for this design arises from cases where financial institutions release only aggregated balance time series, often due to privacy or regulatory constraints. For example, [8] investigates clustering of financial account time series after a bank provided balance data without the corresponding transactions, demonstrating the feasibility of segmentation under such data limitations.

Building on these considerations,this work employs the Sparse QUadratic approximation for Inverse Covariance (SQUIC) and SQUIC-Fit algorithms [9, 10] to estimate a sparse precision

matrix, which encodes conditional dependencies between accounts and enables the construction of a graph structure without requiring direct transactional links. Multiple graph clustering algorithms, such as Spectral Clustering [11], the Louvain [12] and Leiden [13] methods and DBSCAN [14], are then applied to discover customer segments. The proposed methodology generates behaviorally grounded clusters that may help financial institutions enhance risk management, monitoring, and personalization strategies by revealing structurally consistent patterns in the customer base. This is particularly relevant in settings where labeled data are scarce, since annotations are often difficult and costly to obtain from domain experts [15, 16]. These limitations obstruct the use of supervised learning models for customer segmentation, reinforcing the value of unsupervised clustering approaches that divide customers into homogeneous groups based on behavioral patterns encoded in financial time series [17].

1.0.1 Objectives

The main objectives of this research can be summarized as:

- preprocess customer transaction data from synthetic financial datasets into structured time-series formats suitable for downstream analysis;
- apply the SQUIC algorithm to these time-series representations in order to infer a sparse inverse covariance matrix that reflects statistical dependencies between customers;
- construct a weighted graph from the SQUIC output, where nodes represent customers and edges encode estimated behavioral similarities;
- perform graph-based clustering on the resulting structure, identifying distinct customer segments based on behavioral similarity;
- evaluate the quality of the retrieved communities using graph-based clustering metrics, thereby enabling the assessment of their interpretability and their potential value for financial decision-making.

1.0.2 Structure of the thesis

This thesis is organized into five chapters, each addressing a specific aspect of the research. Together, they provide the necessary background, describe the methodological framework, and present and discuss the results.

Chapter 2, introduces the key concepts relevant to this work, including client segmentation approaches, time-series representation and the algorithms employed (SQUIC, SQUIC-Fit and clustering methods). This chapter also describes the datasets and evaluation metrics used throughout the thesis. Chapter 3 outlines the overall research design and explains the experimental setup. It details the two experiments on which the thesis is based, describing the steps taken to preprocess the data, infer network structures, and perform clustering. The numerical results of these experiments are presented in Chapter 4, where findings are explained and illustrated using quantitative metrics and visualizations, with emphasis on the performance and interpretability of the proposed segmentation framework. Chapter 5 summarizes the main contributions of the thesis, reflects on its limitations and suggests directions for future research.

1.0.3 Reproducibility

All code and experiments developed for this work are publicly available in a dedicated repository[1]. The repository includes the Python scripts for the experiments, the implemented functions and the configuration files required to reproduce the results. The specific versions of the Python libraries used are documented in Appendix A.1.

[1] https://github.com/cricci3/FinancialCrimeModels

Chapter 2

Background

2.1 Client segmentation

Customer segmentation groups customers into peer clusters based on shared characteristics, such as industry type, transaction frequency or average volume. Segmentation can be implemented using various analytical approaches. Legacy systems typically rely on manually defined segments, but modern methods employ statistical learning and machine learning techniques to create fine-grained groupings without manual rule engineering. For example, clustering algorithms can partition the customer base into into finer partitions of the feature space based on observed behavioral data.

The benefits of client segmentation are substantial. It enables organizations to personalize offerings, improve targeting precision and enhance overall customer satisfaction. From a strategic perspective, segmentation supports informed decision-making by highlighting the needs of distinct customer groups, guiding product development, marketing campaigns, and customer relationship management. As a result, segmentation has become a cornerstone of modern data-driven business practices and is widely applied across industries ranging from retail and telecommunications to finance and healthcare [18].

2.2 Time series

A time series is a sequence of observations recorded over time on a specific phenomenon, typically measured at regular intervals such as years, months, weeks or days. Unlike cross-sectional data, time series observations possess an inherent temporal ordering. This ordering implies that successive values are not independent; rather, the value at a given point in time is often influenced by past values. Consequently, time series analysis seeks not only to describe the observed data but also to model the temporal dependence that governs its evolution.

Formally, a time series can be represented as a collection of random variables indexed in time [19]:

$$\{X_1, X_2, \ldots, X_T\}, \tag{2.1}$$

where X_t denotes the observation recorded at time t. The realized sequence of values is written as $\{x_1, x_2, \ldots, x_T\}$, with x_t corresponding to the observation at time t. A distinctive feature of time series analysis is its causal structure: models are typically constructed such that the

present depends on past values, reflecting the one-way flow of time, while future values cannot influence the past.

2.2.1 Decomposition

A common approach to understanding the structure of a time series is decomposition, which separates the observed signal into interpretable components. Classical decomposition expresses a time series as the sum of three terms [20]:

$$y_t = T_t + S_t + R_t, \tag{2.2}$$

where y_t is the observed value at time t and:

- Trend T_t represents the long-term progression of the series, capturing its overall direction or underlying movement;

- Seasonality S_t reflects systematic and calendar-related fluctuations that repeat at regular intervals (e.g., daily, weekly, monthly). This aspect is particularly relevant in financial data, where activity often follows weekly cycles (e.g., payroll) or monthly cycles (e.g., bill payments). Such predictable fluctuations are common across many accounts and do not necessarily imply a direct, meaningful relationship between them;

- Residual component R_t represents the remainder of the series after removing trend and seasonality, capturing irregular, random and unexpected fluctuations in the data [21].

Many statistical learning techniques assume that observations are independent and identically distributed (i.i.d.), an assumption that is often violated in time series data. Financial and transactional records, in particular, exhibit strong temporal dependencies, such as autocorrelation, long-term trends and seasonal cycles. If these systematic components are not removed, they may dominate similarity measures and obscure the intrinsic behavioral patterns of interest. For example, clustering algorithms applied directly to raw transaction series risk grouping together customers who share common calendar effects (e.g., monthly salary deposits or weekly bill payments) rather than reflecting meaningful differences in their financial behavior. A simple illustration of decomposition is shown in Figure 2.1, where the observed series is separated into trend, seasonal and residual components. This preprocessing step makes it possible to isolate recurrent cycles from the irregular fluctuations that are most informative for behavioral segmentation.

Decomposition mitigates this problem by isolating trend and seasonal components, leaving behind a residual series that is closer to stationarity and more weakly dependent over time. The residual captures irregular fluctuations, which are often the most informative for identifying distinct behavioral groups. By preprocessing the data in this way, decomposition aligns time series more closely with the assumptions underlying covariance estimation and clustering, thereby improving both the interpretability and reliability of the resulting segmentation [22].

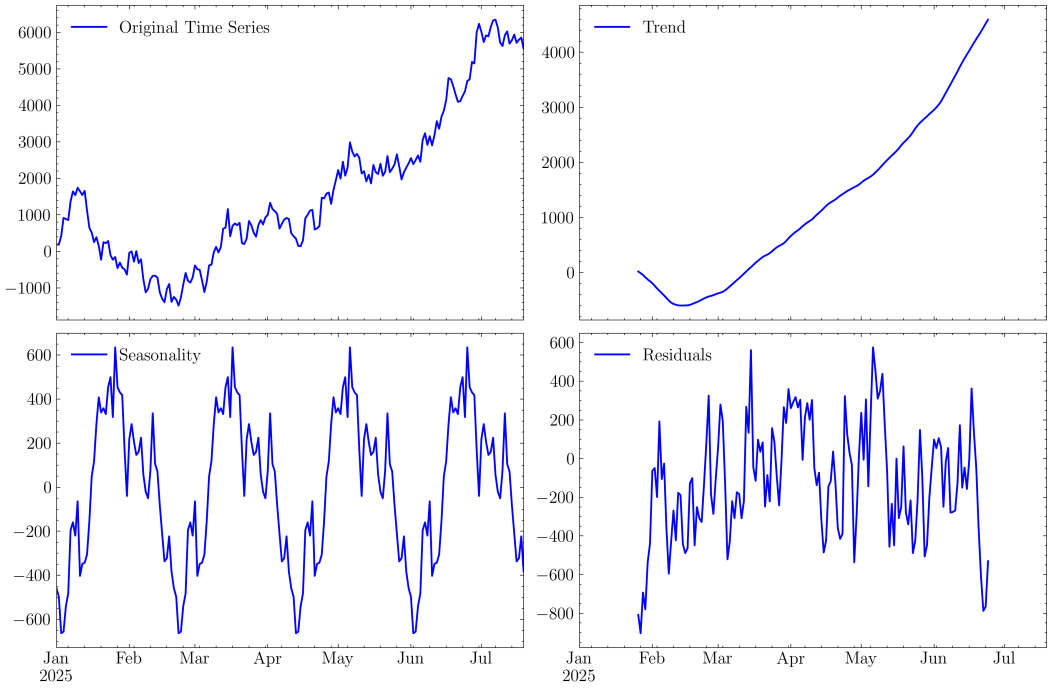

Figure 2.1. Illustrative example of time series decomposition into trend, seasonal and residual components

2.3 Clustering

Clustering is a fundamental unsupervised learning technique aimed at discovering inherent groupings or structures within data without prior label information. Formally, it is the task of partitioning a dataset into a finite set of clusters such that data points within the same cluster are more similar to one another than to those in different clusters. Similarity is typically quantified using distance metrics, with Euclidean, Manhattan and Cosine distances being among the most commonly employed [23].

The clustering problem can be formally defined as follows [24]: given a data set $Z = \{z_1, z_2, ..., z_p, ..., z_n\}$ where z_p is a pattern in the N_d-dimensional feature space and N_p is the number of patterns in Z, then the clustering of Z is the partitioning of Z into K clusters $\{C_1, C_2, \ldots, C_K\}$ satisfying the following conditions:

- each pattern should be assigned to a cluster, i.e. $\cup_{k=1}^{K} C_k = Z$,
- each cluster has at least one pattern assigned to it, i.e. $C_k \neq \emptyset, k = 1, ..., K$,
- each pattern is assigned to one and only one cluster (in case of hard clustering only), i.e. $C_k \cap C_\ell = \emptyset$ where $k \neq \ell$.

Most clustering algorithms can be classified in one of the two most popular techniques, *hierarchical* and *partitional* clustering [25]. Hierarchical clustering algorithms build a tree-like structure (*dendrogram*) by recursively merging or splitting clusters. Agglomerative approaches begin with each point as a separate cluster and iteratively merge the most similar pairs, whereas

divisive methods start with all points in one cluster and split recursively. A key advantage is that the number of clusters need not be predefined, though hierarchical methods are computationally expensive and static (i.e., once merged, clusters cannot be re-split) [23]. Partitional clustering algorithms divide the dataset into a fixed number of clusters by optimizing a certain criteria such as intra-cluster variance. However, these optimization problems are generally *NP-hard* and combinatorial, so, some probabilistic technique for approximating the global optimum of the given function are used [26]. This second type of algorithms are generally more efficient and flexible in reassigning points, but they require the number of clusters to be fixed a priori and may converge to local optima rather than the global solution [23].

Beyond this classical taxonomy, several other families have emerged to address limitations in handling complex structures, large-scale data or, as for our case, networked relationships.

2.3.1 Graph-based clustering

A set of n data points $x_1, \ldots, x_n$ together with a pairwise similarity measure s_{ij} can be naturally represented as a graph $G = (\mathcal{V}, \mathcal{E})$. The vertex set $\mathcal{V} = \{v_1, \ldots, v_n\}$ corresponds to the data points, with each vertex v_i representing a data point x_i. The edge set $\mathcal{E}$ contains edges e_{ij} that encode relationships between pairs of vertices v_i and v_j. The weighted adjacency matrix $W \in \mathbb{R}^{n \times n}$ stores non-negative edge weights $w_{ij} \geq 0$, where w_{ij} quantifies the similarity between v_i and v_j. A value of $w_{ij} = 0$ indicates the absence of a direct connection between the two vertices [27].

Similarity measures can be defined in various ways, such as through a Gaussian kernel applied to the distance between data points, or, as in this study, by constrained inverse covariance matrix estimation [10]. For simple, undirected graphs, self-loops are excluded ($w_{ii} = 0$) and the adjacency matrix is symmetric, satisfying $w_{ij} = w_{ji}$.

Graph-based representations provide a natural foundation for clustering techniques that exploit structural information in the data. Once the data is modeled as a graph, a variety of algorithms can be applied to identify groups of closely related vertices. Among these, *community detection* methods such as Louvain and Leiden aim to maximize a quality function (typically Modularity) to uncover densely connected subgraphs, making them highly effective for large-scale network analysis. Spectral clustering, on the other hand, leverages the eigenvectors of the graph Laplacian to project the data into a lower-dimensional embedding space, where clusters can be more easily separated. While these methods are inherently graph-based, other clustering paradigms can also benefit from the graph representation; for instance, *density-based* approaches such as DBSCAN can operate on similarity graphs to identify high-density regions and detect noise.

The following subsections provide a detailed description of the four clustering algorithms employed in this work: Leiden, Louvain, Spectral clustering and DBSCAN.

2.3.1.1 Louvain method

The Louvain method [12] is a greedy optimization algorithm for detecting communities in networks. It aims to maximize the modularity of a partition, a measure of the density of links inside communities compared to links between them.

Modularity, with values in the in the range $[-1, 1]$, is defined for weighted graphs as:

$$Q = \frac{1}{2m} \sum_{i=1}^{N} \sum_{j=1}^{N} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \tag{2.3}$$

where A_{ij} denotes the edge weight between nodes i and j, k_i and k_j are the corresponding weighted degrees of these nodes and m represents the total sum of all edge weights in the graph. The terms c_i and c_j indicate the community assignments of nodes i and j, respectively. Last, $\delta(c_i, c_j)$ is the Kronecker delta function, which equals 1 if $c_i = c_j$ and 0 otherwise.

The Louvain algorithm proceeds iteratively in two main phases. In the first phase, called Local Optimization, initially each node is assigned to its own community (Figure 2.2). Then, for each node v, the algorithm considers moving it to the community of each of its neighbors. It chooses the move that maximally increases modularity. If no move increases modularity, the node remains in its current community. This is repeated for all nodes until no further modularity gain is possible (Figure 2.3). In the second phase, Community Aggregation, communities identified in the previous step are collapsed into single nodes. Edges between communities become weighted edges in the new graph and self-loops represent intra-community connections. This forms a new, smaller graph (Figure 2.4), to which the first phase can again be applied. These two phases can be repeated so that more nodes are moved into existing communities until an optimal level of modularity is reached.

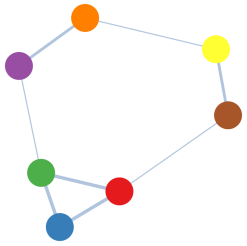

Figure 2.2. Louvain Phase 0

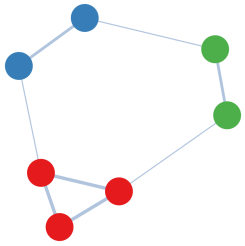

Figure 2.3. Louvain Phase 1

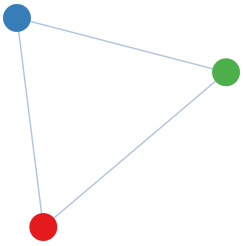

Figure 2.4. Louvain Phase 2

2.3.1.2 Leiden algorithm

The Louvain algorithm has gained popularity due to its speed and high-quality performance in comparative studies, and it remains one of the most frequently cited methods in the field of community detection. However, despite its success, Louvain suffers from a critical limitation: it can produce communities that are internally disconnected or only weakly connected, which undermines the interpretability of the resulting partitions.

To address this issue, the Leiden algorithm [13] was introduced as an improved alternative. It not only guarantees that all communities are well connected but also achieves higher-quality partitions with improved convergence properties. Moreover, Leiden builds on earlier enhancements such as the smart local move, fast local move and random neighbor move strategies.

The Leiden algorithm operates in three main phases. The first phase is the local movement of nodes, where nodes are relocated between communities to maximize modularity (Equation 2.3). In the second phase, the partition is refined by splitting communities into subcommunities until all are internally connected. This refinement step prevents the occurrence of disconnected components within the same community, a flaw that may arise in Louvain. Then, in the third

phase, the network is aggregated based on the refined partition, and the process is repeated iteratively on the coarser graph.

To formalize these steps, the pseudocode in Algorithm 1 summarizes the main operations of the Leiden method. Compared to Louvain, the distinguishing feature is the refinement step (`get_p_refined`), which ensures that communities remain internally connected before aggregation. In this phase, each node is temporarily placed in its own singleton community and then iteratively reassigned to neighboring communities in order to maximize modularity. After every reassignment, the corresponding community is refined to prevent disconnected substructures. This refinement procedure produces the intermediate partition P_{refined}, which is subsequently used to construct the aggregated graph for the next iteration.

Algorithm 1: Leiden community detection [13]

Input: Graph $G = (V, E)$, initial partition P

repeat
  $P \leftarrow$ `louvain_move_nodes`(G, P) ; `// Relocate nodes to maximize modularity`
  if $|P| = |V|$ **then**
    `done` $\leftarrow$ True ;
  else
    $P_{\text{refined}} \leftarrow$ `get_p_refined`(G, P) ; `// Refine communities to ensure internal connectivity`
    $G \leftarrow$ `aggregate_graph`(G, P_{refined}) ; `// Aggregate refined communities into single nodes`
    $P \leftarrow \{\{v \mid v \subseteq C, v \in V(G)\} \mid C \in P\}$;
until *done*;

Output: Final partition P of V

2.3.1.3 Spectral Clustering

Spectral clustering is a technique grounded in multivariate statistics and graph theory. It leverages the eigenvalues and eigenvectors of the Laplacian matrix derived from a similarity graph constructed over the data. By embedding the data into a lower-dimensional subspace defined by the first k eigenvectors, the algorithm transforms the clustering problem into one where distance-based methods can be applied effectively. In particular, k-means is typically employed in this spectral embedding space to obtain discrete partitions from the continuous eigenvector representation. The similarity matrix, often derived from a graph adjacency matrix, encodes pairwise similarities between data points [11].

Each vertex in the graph received as input, represents a data point and weighted edges encode pairwise similarities. From the weighted adjacency matrix W, the degree matrix D is formed, where each diagonal entry d_i is the sum of edge weights incident to vertex v_i. The unnormalized graph Laplacian is then defined as $L = D - W$, a symmetric positive semi-definite matrix that encodes the graph's connectivity structure and its weights. A commonly used normalized variant adjust for variations in vertex degree to mitigate bias from highly connected nodes is the symmetric normalized Laplacian:

$$L_{norm} = I - D^{-1/2} W D^{-1/2}, \tag{2.4}$$

where $I \in \mathbb{R}^{n \times n}$ is the identity matrix [27]. The normalized Laplacian has been shown to be more effective in identifying clusters, particularly in arbitrarily weighted or irregular networks,

as it ensures that clustering results are less influenced by nodes with disproportionately high degree [28].
Spectral clustering proceeds by solving an eigenvalue problem on one of these Laplacians. In the ideal case of k disconnected components, the first k eigenvectors are exactly the cluster indicator vectors [11]. When the graph is nearly separable, perturbation theory guarantees that these eigenvectors remain close to the ideal indicator vectors, thus preserving cluster structure. The embedding step uses the first k eigenvectors (corresponding to the smallest eigenvalues) as new coordinates, mapping the data into a low-dimensional Euclidean space where clusters become more separable. As last step, a distance-based clustering method, most commonly *k-means*, is applied in this spectral embedding space to obtain the discrete partition. In this formulation, the eigenvectors serve a dual role: they encode the global connectivity structure of the graph while simultaneously providing an embedding in which conventional clustering algorithms can operate effectively, even in the presence of non-convex or irregularly shaped clusters [11].

Algorithm 2: Spectral Clustering Algorithm [11]

Input: Adjaceny Matrix A
1 Calculate the Laplacian L or normalized Laplacian L_{norm};
2 Calculate the first k eigenvectors (the eigenvectors corresponding to the k smallest eigenvalues of L);
3 Consider the matrix formed by the first k eigenvectors; the $l-th$ row defines the features of graph node l;
4 Cluster the graph nodes based on these features (e.g., using k-means clustering);
Output: Partition given by clustering

2.3.1.4 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [14] is a non-parametric clustering algorithm that defines clusters as regions of high point density, separating them from sparse regions considered as noise. DBSCAN classifies points into the following categories:

- a point p is a *core point* if at least $minPts$ points (including p) are within distance ϵ of it. An example is point A in Figure 2.6;
- a point q is *directly reachable* from p if q lies within distance ϵ from a core point p. All light-blue points in Figure 2.6 are examples;
- a point q is *reachable* from p if there exists a chain of directly reachable points connecting p to q. In Figure 2.6, Points C and D are two examples of this type of points, in fact they are not directly reachable from A but are transitively reachable from core point A through point B;
- points that are not reachable from any core point are considered *noise* or *outliers*. For example, point N in Figure 2.6.

If p is a core point, it forms a cluster together with all points (core or border) reachable from it (each cluster contains at least one core point); consequently, each cluster contains at least one core point. In Algorithm 6, the `ExpandCluster` subroutine performs this expansion by iteratively labeling all points connected to p through chains of density-reachability, thus growing the cluster as illustrated in Figure 2.7.

Unlike partition-based clustering methods, DBSCAN does not require the number of clusters to be specified before and can detect clusters of arbitrary shape. It operates using global parameters ϵ and *minPts*, ideally chosen to capture the least dense meaningful cluster in the data.

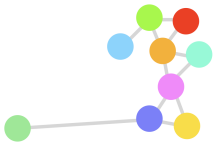

Figure 2.5. DBSCAN Phase 0

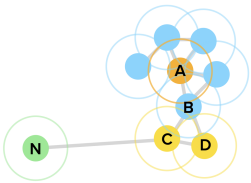

Figure 2.6. DBSCAN Phase 1

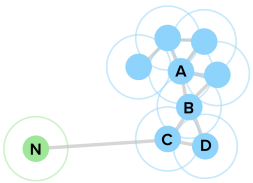

Figure 2.7. DBSCAN Phase 2

Algorithm 3: DBSCAN Algorithm [14]

Input: Set of points $\mathcal{D}$, neighborhood radius ε, minimum number of points $minPts$

```
Initialize all points in 𝒟 as UNCLASSIFIED ;                    // Figure 2.5
clusterId ← 0;
foreach point p in 𝒟 do
    if p is UNCLASSIFIED then
        if ExpandCluster(𝒟, p, clusterId, ε, minPts) then
            clusterId ← clusterId +1;
```

Output: Clusters of density-connected points, labeled in $\mathcal{D}$

2.3.2 Clustering evaluation

Clustering quality can be assessed using two main approaches: *external* evaluation, when ground-truth labels are available, and *internal* evaluation, when such labels are absent.

2.3.3 External metrics

External metrics compare the predicted cluster partition against a known reference partition, measuring how closely the discovered clusters match the ground-truth structure. Common examples include the Adjusted Rand Index (ARI) and the F1-Score.

2.3.3.1 F1-score

The F1-Score [29] is the harmonic mean of precision and recall, providing a balanced measure of a classifier's ability to correctly identify positive instances while avoiding false positives and false negatives. It is defined as:

$$F_1 = \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}} \tag{2.5}$$

where:

$$\text{precision} = \frac{TP}{TP + FP}, \text{recall} = \frac{TP}{TP + FN} \tag{2.6}$$

and TP, FP, and FN denote true positives, false positives, and false negatives, respectively.

The highest possible value of the F1-score is 1, which indicates both perfect precision and recall, corresponding to a perfect retrieval of the provided labels from the clusters. Against, the lowest possible value is 0, which occurs when either precision or recall equals zero.

2.3.3.2 ARI (Adjusted Rand Index)

The Adjusted Rand Index [30] is widely used for validating clustering results against ground-truth partitions. It computes similarity between discovered communities and "ground-truth" communities. ARI is defined as follows:

$$ARI = \frac{\sum_{ij}\binom{n_{x_i y_j}}{2} - \left[\sum_i \binom{n_{x_i}}{2}\sum_j \binom{n_{y_j}}{2}\right] \Big/ \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{n_{x_i}}{2} + \sum_j \binom{n_{y_j}}{2}\right] - \left[\sum_i \binom{n_{x_i}}{2}\sum_j \binom{n_{y_j}}{2}\right] \Big/ \binom{n}{2}} \tag{2.7}$$

where, $X = \{x_1, x_2, \ldots, x_i\}$ represents set of detected communities, $Y = \{y_1, y_2, \ldots, y_j\}$ represents set of "ground-truth" communities, n represents total number of nodes, $n_{x_i} = |x_i|$, and $n_{x_i y_j} = |x_i \cap y_i|$.

ARI is a symmetric measure that ranges from -1, when both the communities are totally different, to $+1$, when both the communities are completely similar.

ARI is a symmetric measure that ranges from -1 to $+1$. A value of $+1$ indicates a perfect match between the retrieved clusters and the ground-truth labels, a value close to 0 corresponds to a random assignment, and negative values indicate agreement less than expected by chance.

2.3.4 Internal metrics

Internal metrics, in contrast, quantify the structural quality of the detected communities without relying on ground-truth labels. Examples include Modularity (described in Section 2.3.1.1) and Partition Density.

2.3.4.1 Partition density

Partition density [31] evaluates the compactness of communities, penalising fragmented or sparse clusters. It is defined as:

$$D = \frac{2}{m}\sum_{\alpha=1}^{c} \frac{m_\alpha - (n_\alpha - 1)}{(n_\alpha - 2)(n_\alpha - 1)} \tag{2.8}$$

where, m is the total number of edges, m_α and n_α are the numbers of edges and vertices in the community α, and c is the number of communities. Note that the metric is defined only for communities with at least three nodes. A value of $D = 0$ indicates that all clusters are too small to contribute to the density measure, while $D = 1$ represents the ideal scenario in which every cluster is internally a clique, i.e., fully connected among its vertices.

This metric is particularly well suited for the evaluation of unsupervised clustering tasks, as it does not rely on external ground-truth labels. It measures the internal cohesiveness of communities by quantifying the density of connections within clusters. Higher values correspond to partitions where communities are internally well connected, which increases the likelihood that the detected clusters reflect meaningful structural patterns in the underlying network.

2.4 Inverse covariance matrix

In multivariate statistics, the *precision matrix* is the inverse of the covariance matrix of a random vector. Let $\mathbf{X} \in \mathbb{R}^p$ have mean $\boldsymbol{\mu}$ and covariance

$$\Sigma = \mathbb{E}\big[(\mathbf{X}-\boldsymbol{\mu})(\mathbf{X}-\boldsymbol{\mu})^\top\big] \in \mathbb{R}^{p\times p}. \tag{2.9}$$

If Σ is positive definite, its inverse $\Theta = \Sigma^{-1}$ is the *precision matrix*.

In Gaussian graphical models, zeros in Θ encode conditional independence: $\Theta_{ij} = 0$ if and only if $X_i \perp X_j \mid X_{\setminus\{i,j\}}$. Moreover, the (pairwise) partial correlation between X_i and X_j given all remaining variables is

$$\rho_{ij|\text{rest}} = -\frac{\Theta_{ij}}{\sqrt{\Theta_{ii}\,\Theta_{jj}}}. \tag{2.10}$$

Thus the sign of $-\Theta_{ij}$ matches the sign of the partial correlation, and larger $|\Theta_{ij}|$ indicates stronger conditional dependence.

Conditional independence for two random vectors $\mathbf{X} = (X_1, \ldots, X_\ell)^\top$ and $\mathbf{Y} = (Y_1, \ldots, Y_m)^\top$ is defined as follows. $\mathbf{X}$ and $\mathbf{Y}$ are *conditionally independent* given $\mathbf{Z} = (Z_1, \ldots, Z_n)^\top$ if [32]

$$(\mathbf{X} \perp\!\!\!\perp \mathbf{Y}) \mid \mathbf{Z} \iff F_{\mathbf{X},\mathbf{Y}|\mathbf{Z}=\mathbf{z}}(\mathbf{x},\mathbf{y}) = F_{\mathbf{X}|\mathbf{Z}=\mathbf{z}}(\mathbf{x})\, F_{\mathbf{Y}|\mathbf{Z}=\mathbf{z}}(\mathbf{y}), \quad \forall\, \mathbf{x},\mathbf{y},\mathbf{z}. \tag{2.11}$$

Under Gaussianity, this definition coincides with the zero–nonzero pattern of Θ, which therefore serves as the adjacency matrix of the associated undirected graphical model.

As noted in Section 2.2.1, account-balance time series contain trend and seasonal components that violate the i.i.d. assumption underlying Gaussian likelihood–based precision estimation. To avoid spurious correlations from shared calendar effects (e.g., payroll or bill-payment cycles), we decompose each series and estimate covariance and precision from the residuals only. This centers inference on irregular co-movements rather than predictable periodicities, yielding a more faithful conditional-dependence structure.

2.4.1 Graphical Lasso

Estimating the precision matrix $\Theta = \Sigma^{-1}$ directly from data is challenging in high-dimensional settings, where the number of variables p may be comparable to or exceed the number of observations n. In such cases, the sample covariance matrix S is often ill-conditioned or singular, making its inversion unstable or impossible. To address this issue, sparse estimation techniques are employed under the assumption that most variables are conditionally independent, so that Θ is sparse.

A widely used approach is the *Graphical Lasso* (glasso) [33], which formulates the estimation problem as the ℓ_1-regularized Gaussian maximum likelihood optimization:

$$\hat{\Theta} = \arg\min_{\Theta \succ 0} \big\{ -\log\det(\Theta) + \operatorname{tr}(S\Theta) + \lambda \|\Theta\|_1 \big\}, \tag{2.12}$$

where S is the sample covariance matrix, $\lambda > 0$ is a regularization parameter, and $\|\Theta\|_1$ denotes the sum of absolute values of the entries of Θ. The ℓ_1 penalty promotes sparsity by shrinking small entries of Θ toward zero, thereby identifying the most relevant conditional dependencies. The imposed sparsity is not only statistically meaningful but also computationally beneficial,

enabling the use of high-performance sparse linear-algebra routines and large-scale second-order solvers.

Glasso is also expressed in a slightly more general form where the scalar λ is replaced by a matrix of entrywise penalties $\Lambda \in \mathbb{R}^{p\times p}$:

$$\hat{\Theta} = \arg\min_{\Theta\succ 0} \{-\log\det(\Theta) + \mathrm{tr}(S\Theta) + \|\Lambda \odot \Theta\|_1\}, \tag{2.13}$$

where $\odot$ denotes the Hadamard (elementwise) product. The scalar formulation (Equation 2.12) is recovered by setting $\Lambda_{ij} = \lambda$ for all entries. This generalization allows for entry-specific penalization and the incorporation of prior structural information.

Although glasso provides a convex optimization framework with strong theoretical guarantees, its scalability is limited in very high-dimensional problems. To address this, second-order methods such as QUIC [34] and BigQUIC [35] were developed, significantly accelerating convergence and enabling problems with tens of thousands of variables. Building upon these ideas, SQUIC [9] further extends the approach to even larger problem sizes.

2.4.2 Sparse QUadratic approximation of Inverse Covariance (SQUIC)

The Sparse Quadratic approximation of Inverse Covariance (SQUIC) algorithm is a large-scale, second-order method for solving the generalized glasso problem (Equation 2.13). Its key innovation lies in the use of sparse matrix representations and block-oriented computations, which allow it to scale efficiently to problems with hundreds of thousands or even millions of variables.

Given a dataset $Y \in \mathbb{R}^{p\times n}$ consisting of p random variables and n samples, SQUIC produces both a sparse estimate of the precision matrix $\hat{\Theta}$ and its sparse approximate inverse $\hat{\Theta}_{\text{inv}}$ (the covariance matrix). The algorithm employs a matrix-valued regularization parameter Λ, which can be specified in flexible ways. A common construction is:

$$\Lambda_{ij} = \begin{cases} W_{ij} & \text{if } W_{ij} \neq 0, \\ \lambda & \text{otherwise,} \end{cases} \tag{2.14}$$

where $W \in \mathbb{R}^{p\times p}$ is a sparse symmetric matrix encoding prior structural information, and $\lambda > 0$ is a global tuning parameter. Optionally, a smaller constant $\eta \in (0, \lambda]$ may be used to relax the penalty on selected entries of W. In this way, SQUIC generalizes the scalar-penalized glasso while retaining scalability [36].

2.4.3 SQUIC-Fit

SQUIC-Fit [10] is an extension of the SQUIC [9] algorithm designed for the efficient estimation of M-matrices, i.e., symmetric positive-definite matrices with non-positive off-diagonal elements.

In the context of Gaussian graphical models, this structure corresponds to an attractive Gaussian Markov Random Field, where all partial correlations between variables are non-negative [37]. This follows from the fact that the partial correlation between variables i and j is proportional to $-\Theta_{ij}$; so, $\Theta_{ij} \leq 0$ implies a non-negative partial correlation. This property ensures that the graph encoded by the precision matrix has only non-negative edge weights, which is particularly important when constructing graph representation for clustering, since it guarantees positive semi-definiteness and facilitates meaningful interpretation of connectivity patterns.

SQUIC-Fit adopts a two-stage approach. It leverages SQUIC to estimate precision matrices twice in succession, using the first estimate to construct a graphical bias that guides the second estimation toward an M-matrix structure. The final M-matrix is obtained via a post-processing step.

Given a dataset $Y \in \mathbb{R}^{p\times n}$, with p variables and n samples, the algorithm is as follows:

- first SQUIC estimation ($\hat{\Theta}^{(1)}$): solve the generalized glasso problem (Equation 2.12) with a scalar penalty parameter $\lambda > 0$. The value of λ is selected to produce a sufficiently sparse initial graph.
- graphical bias extraction: identify pairs of variables with negative off-diagonal entries in $\hat{\Theta}^{(1)}$ whose magnitude exceeds a small threshold $\kappa > 0$. These indicate positive conditional correlations in the underlying underlying Gaussian graphical model. Form the binary adjacency-like matrix:

$$G_{ij} = \begin{cases} 0 & \text{if } i = j, \\ \mathbb{I}(-\hat{\Theta}^{(1)}_{ij} > \kappa) & \text{if } i \neq j \end{cases} \tag{2.15}$$

  where $\mathbb{I}(\cdot)$ is the indicator function.
- matrix-valued regularization parameter: construct a matrix tuning parameter $\Lambda \in \mathbb{R}^{p\times p}$ using:

$$\Lambda_{ij} = \begin{cases} \eta & \text{if } G_{ij} \neq 0, \\ \lambda & \text{if } G_{ij} = 0, \end{cases} \tag{2.16}$$

  where $\eta \in (0, \lambda]$. This penalizes entries outside the detected bias structure more heavily, steering the solution toward the M-matrix form while retaining flexibility within G.
- second SQUIC estimation ($\hat{\Theta}^{(2)}$): solve the ℓ_1-regularized problem again, but now with the matrix-valued penalty Λ (Equation 2.13).
- post-processing to enforce M-matrix form: retain only the negative off-diagonal entries of $\hat{\Theta}^{(2)}$ whose magnitude exceeds κ; set all other off-diagonal entries to zero. The diagonal entries remain unchanged:

$$\hat{\Theta}_{ij} = \begin{cases} 0 & \text{if } i = j, \\ \mathbb{I}(-\hat{\Theta}^{(2)}_{ij} > \kappa) \cdot \hat{\Theta}^{(2)}_{ij} & \text{if } i \neq j, \end{cases} \tag{2.17}$$

  The resulting $\hat{\Theta}$ is symmetric positive-definite and satisfies the M-matrix structure.

The algorithmic steps of SQUIC-Fit are summarized in Algorithm 4.

Algorithm 4: SQUIC-Fit [10]

Input: Data matrix $Y \in \mathbb{R}^{p\times n}$, scalar penalty $\lambda > 0$, reduced penalty $\eta \in (0, \lambda)$, threshold $\kappa > 0$, convergence tolerance $\tau > 0$

1 Estimate : $\hat{\Theta}^{(1)}$; // Eq. 2.12
2 Build graphical bias **G** ; // Eq. 2.15
3 Build matrix regularization parameter Λ ; // Eq. 2.16
4 Estimate : $\hat{\Theta}^{(2)}$; // Eq. 2.13
5 Build M-matrix $\hat{\Theta}$; // Eq. 2.17

Output: Estimated M-matrix $\hat{\Theta}$

From M-matrix to adjacency. As explained above, SQUIC-Fit returns an M-matrix precision estimate $\hat{\Theta}$, that for Equation (2.10), entails nonnegative partial dependencies, since $-\frac{\hat{\Theta}_{ij}}{\sqrt{\hat{\Theta}_{ii}\hat{\Theta}_{jj}}} \geq 0$ for $i \neq j$. We therefore interpret $\hat{\Theta}$ as a nonnegative weighted adjacency by mapping

$$w_{ij} = \mathbb{I}\big(-\hat{\Theta}_{ij} > \kappa\big)(-\hat{\Theta}_{ij}), \qquad w_{ii} = 0 \tag{2.18}$$

As results, nonzero w_{ij} give the edge weight between nodes i and j, while zeros correspond to missing edges. This yields a simple, undirected graph.

2.5 Datasets

The scarcity of publicly available datasets for financial transaction analysis remains a significant challenge for the research community [38]. Due to the intrinsically private nature of financial data, institutions are generally unable to share detailed transaction records, resulting in a lack of legitimate datasets for open research. To address this limitation, synthetic data generators and simulators have emerged as a viable alternative, enabling the creation of realistic, large-scale datasets without compromising customer privacy. In this study, two such simulators were selected: AMLSim [39], which models regulated banking transactions with embedded AML typologies and PaySim [38], which simulates mobile money ecosystems calibrated from real-world data.

2.5.1 AMLSim

AMLSim is a multi-agent simulation platform developed to generate synthetic datasets for AML research [40]. It models a financial transaction environment in which each agent represents a bank account capable of sending funds to other accounts.

The AMLSim architecture consists of two main components, the Transaction Graph Generator and the Transaction Simulator. The first one, creates the overall transaction network structure in Python, defining accounts with attributes (e.g., country, business type) and constructing a base graph from a degree distribution parameter file. The second, is a multi-agent simulation engine that iterates through discrete time steps. At each step, account agents initiate transactions to their neighbors according to the generated graph.

The simulation then, proceeds in three main stages. During *initiation*, the system loads configuration parameters, generates the transaction graph, assigns account attributes and embeds the alert patterns. In the *execution* stage, agents perform transactions over time following the graph structure, thereby generating both normal and suspicious flows. Finally, in the *finalization* stage, the simulator outputs several CSV files; of particular importance to this work are the *accounts* file, which lists user IDs, initial balances, and fraud labels, and the *transactions* file, which records the timestamp, sender and receiver IDs, transaction amount, and an indicator of whether the transaction is fraudulent.

In our numerical experiments (Section 3.0.1), we consider the following publicly available AMLSim datasets:

- *AMLSim100*: 100 clients and 10 000 transactions;
- *AMLSim1K*: 1 000 clients and 100 000 transactions;

- *AMLSim10K*: 10 000 clients and 1 000 000 transactions;
- *AMLSim100K*: 100 000 clients and 10 000 000 transactions.

2.5.2 PaySim

PaySim [38] is a financial simulator that simulates mobile money transactions based on an original dataset, logs extracted from a mobile money service implemented in an African country and provided by Ericsson[1] to the paper's authors. PaySim uses a MABS (Multi-Agent Based Simulation) approach to model clients, merchants and banks as autonomous agents, each with specific transaction limits, profiles and behaviors. Through the application of statistical analysis and social network analysis on the original dataset, PaySim is able to simulate key transaction types (*Cash-In*, *Cash-Out*, *Debit*, *Payment* and *Transfer*) and produce synthetic data that closely resembles the statistical properties of the real data.

PaySim's design follows the *Overview, Design concepts, Details* (ODD) modeling protocol [41], and can be described in two main components: Statistical Transaction Profile Generator and Multi-Agent Transaction Simulator. Statistical Transaction Profile Generator extracts transaction type distributions, frequencies and amount statistics from the original dataset and it also incorporates social network analysis to determine agent connectivity and activity rates. Multi-Agent Transaction Simulator, instead, implements the agent interactions using the MASON toolkit [42]. Define three types of Agents: Clients, Merchants and Banks, that execute transactions over discrete time steps according to calibrated probability distributions and operational constraints.

The simulation unfolds in three sequential stages. During the *initiation* stage, PaySim loads parameters (e.g., simulation seed, input/output file paths), aggregated transaction statistics, initial agent balances, and the maximum number of allowed transaction repetitions. In the *execution* stage, agents act at each time step (day/hour) by probabilistically selecting a transaction type and partner. Supported transaction types are:

- *Cash-In*: a certain amount is exchanged from a Merchant to a Client,
- *Cash-Out*: a certain amount is exchanged between two users,
- *Debit*: a certain amount is exchanged from a Client to a Bank,
- *Payment*: a certain amount is exchanged from a Client to a Merchant,
- *Transfer*: a certain amount is exchanged from a Client to a Client,

Transaction amounts are drawn from normal distributions parameterized by the mean and variance for that transaction type,

$$\text{Amount} \sim \mathcal{N}(\mu, \sigma^2) \tag{2.19}$$

where μ is the mean transaction amount and σ^2 its variance. The realism of the generated data is evaluated by comparing it to the original dataset using the Sum of Squared Errors (SSE). Finally, in the *finalization* stage, the simulator outputs a CSV file in which each record contains the transaction timestamp, amount exchanged, sender and receiver IDs (prefixed by user type: C for Client, M for Merchant, B for Bank), pre- and post-transaction balances.

[1] `https://www.ericsson.com/en/about-us`

Since the authors of PaySim do not provide pre-generated datasets, we used the simulator to generate datasets at nominal scales comparable to AMLSim (100, 1K, 10K, 100K). Due to the stochastic of PaySim, the realized numbers of clients and transactions can deviate slightly from the user-specified targets; we therefore report realized counts. In our case:

- *PaySim100*: 111 clients and 12 492 transactions;
- *PaySim1K*: 1 026 clients and 103 884 transactions;
- *PaySim10K*: 10 284 clients and 1 100 726 transactions;
- *PaySim100K*: 102 249 clients and 10 900 690 transactions.

A key characteristic of the datasets generated by AMLSim and PaySim is that the transaction records and the derived financial time series are not independent. Since account amounts evolve as the cumulative result of transactions, the activity of one account is directly linked to that of another whenever they interact. This coupling produces synchronized patterns in the time series and violates the assumption of independence between samples.

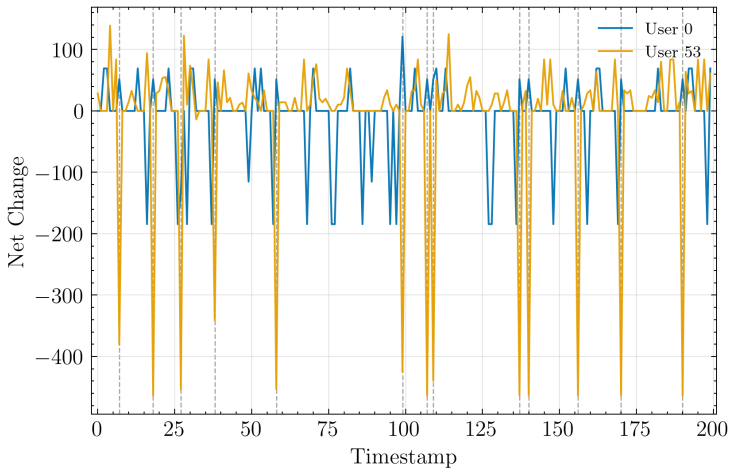

Figure 2.8. Net-Change time series of users 0 and 53

For example, in the AMLSim dataset with 100 accounts, *users 0* and *53* are involved in 13 transactions, primarily transfers from account *53* to account *0*. Each transfer produces a simultaneous decrease in account 53's balance and a corresponding increase in account 0's balance (Figure 2.8). The magnitudes of these changes differ, as account 53 is also transacting with other users on the same days, but the correlation between the two series remains evident. Such synchrony illustrates the dependent nature of the data and demonstrates why standard i.i.d. assumptions do not hold. This motivates the use of time-series decomposition to attenuate dependencies and better isolate intrinsic behavioral patterns.

Chapter 3

Client segmentation with graph clustering

This chapter presents the methodological framework for client segmentation on two synthetic datasets, PaySim and AMLSim, introduced in Sections 2.5.1– 2.5.2. We transform raw transaction records into daily account-balance time series to obtain a representation that abstracts from transaction-level details and emphasizes the temporal dynamics of balances, reflecting scenarios in which only account-level aggregates are available rather than detailed logs. Each user u_i is represented by

$$[b_{i,1}, b_{i,2}, \ldots, b_{i,T}], \tag{3.1}$$

where $b_{i,t}$ denotes the account balance on day t and T is the length of the observation window.

The methodology is divided in two experiments. A common processing pipeline is applied to both datasets. This pipeline transforms raw transactions into user time series, preprocesses the data via decomposition and normalization, estimates a sparse precision matrix through the SQUIC-Fit algorithm and derives an interaction graph. On this graph, different clustering algorithms, Louvain, Leiden, DBSCAN and Spectral Clustering, are applied to explore alternative partition structures, and the resulting clusters are evaluated using internal metrics such as modularity and partition density (Section 2.3.4). The overall workflow is summarized in Algorithm 5.

An additional experiment is conducted on the PaySim datasets. Unlike AMLSim, PaySim provides explicit user labels distinguishing clients (C) from merchants (M) (banks are excluded from the analysis, as each dataset contains only one). These labels enable a supervised validation study. Specifically, Spectral Clustering with $k = 2$ is applied to the inferred graph structure from SQUIC-Fit in order to test whether the learned representations can effectively separate clients from merchants. The workflow of this second experiment is summarized in Algorithm 6. This complementary study serves as a means to interpret the clustering results in a setting where ground-truth information is available.

The remainder of this chapter is structured as follows: Section 3.0.1 presents the common pipeline shared by both datasets, detailing the preprocessing, graph learning, clustering and evaluation stages; Section 3.0.2 describes the PaySim-specific supervised study.

3.0.1 AMLSim and PaySim: common experiment

Algorithm 5: Common Processing Pipeline

Input : Target dataset **name** and **dimension** dim (e.g., 100, 1K, 10K, 100K)

1 **Data acquisition and preprocessing**;
2 `transactions` ← Load `transactions.csv` file corresponding to dim and $name$;
3 Y ←Build users × days balance table $Y \in \mathbb{R}^{p \times n}$ from `transactions`;
4 $Y_{residual}$ ← DECOMPOSE(Y);
5 Y ← NORMALIZE($Y_{residual}$) ;
6 M**-matrix estimation**;
7 λ ← Read reg. parameters associated to dataset from configuration JSON;
8 (Θ, NNZ, NNZ\ROW, TIME) ← SQUIC_FIT(Y, λ);
9 VISUALINTERPRETATION(Θ);
10 G ← MMATRIXTOGRAPH(Θ) ; `// edge if` $\Theta_{ij} \neq 0$, `weight` $|\Theta_{ij}|$
11 COSMOGRAPHVISUALIZE(G);
12 **Community detection**;
13 **foreach** *method* ∈ *{Spectral, Louvain, Leiden, DBSCAN}* **do**
14 PARTITIONS[$method$] ← COMPUTECLUSTERING($G, method$);
15 **Clustering evaluation**;
16 (Q, PDENSITY) ← EVALUATECLUSTERING(G, PARTITIONS);
17 PLOTRESULTS(G, Q, PDENSITY);
18 COSMOGRAPHVISUALIZE(G, PARTITIONS);
19 **End**;

3.0.1.1 Data acquisition and preprocessing

The process begins (Line 2 of Algorithm 5) with the specification of the dataset name (PaySim or AMLSim) and its dimension (e.g., 100, 1K, 10K, 100K). This user input determines the selection of the corresponding CSV file containing the transactions. In both datasets, each row of the CSV file represents a single transaction, recording the sender, the receiver and the transferred amount. As shown in Line 3 of Algorithm 5, the transaction file is then transformed into a user-day balance matrix $Y \in \mathbb{R}^{p \times n}$, where p denotes the number of users (defined by the chosen dataset size) and n the number of time steps (days). In this representation, each row corresponds to a user, each column to a day and each entry $y_{i,t}$ records the balance of user u_i at time t. This construction gives a compact and structured view of user behavior over time, while deliberately excluding transaction-level features. By focusing solely on temporal balance dynamics, the analysis simulates scenarios in which individual transaction records are unavailable, and only aggregated balance information can be observed.

Following data acquisition, each user–day balance trajectory is decomposed into interpretable components (Line 4 of Algorithm 5). As discussed in Section 2.2.1, classical seasonal–trend decomposition expresses a time series as the sum of trend, seasonality and residual variation (Equation 2.2). Decomposition[1] is applied using with weekly seasonality (`period=7`). With that, the trend $\hat{T}_t$ is first estimated via a centered moving-average (convolution) filter applied

[1] `https://www.statsmodels.org/dev/generated/statsmodels.tsa.seasonal.seasonal_decompose.html`

to y_t. The series is then de-trended, and the seasonal component $\hat{S}_t$ is obtained by averaging the de-trended values at the same within-period positions (here, day-of-week for `period=7`). The remainder, $\hat{R}_t = y_t - \hat{T}_t - \hat{S}_t$, is retained for further analysis, as it captures short-term fluctuations after removing systematic trends and seasonal effects. This choice emphasizes irregular balance dynamics, which are expected to better reflect user-specific behaviors.

The residual matrix $Y_{\text{res}} \in \mathbb{R}^{p \times n}$ obtained from decomposition is then standardized to remove magnitude effects and enable meaningful comparison across users. Normalization (Line 5 Algorithm 5) is performed independently for each user's residual series:

$$y_{i,t} = \frac{y_{\text{res}_{i,t}}}{\sigma_i}, \tag{3.2}$$

where $\sigma_i = \text{std}(y_{\text{res}_{i,:}})$ is the standard deviation of user u_i's residual sequence, constrained to be at least 1 to avoid division by zero. This transformation ensures that all users are represented on a comparable scale, so that the next steps are driven by the shape of temporal dynamics rather than differences in absolute balance levels. For notational simplicity the preprocessed residual–normalized balance matrix will again be denoted as Y in the subsequent discussion, even though Y originally referred to the raw user–day balance matrix.

3.0.1.2 M-matrix estimation

The next stage is the computation of the M-matrix using the SQUIC-Fit algorithm (see Sections 2.4.2– 2.4.3). This step is crucial, as the M-matrix provides a sparse estimate of the inverse covariance structure, which directly encodes conditional dependencies between users.

The SQUIC algorithm requires a regularization parameter λ, which controls the sparsity of the estimated matrix. To account for its influence, a small set of candidate values (typically four or five) is predefined for each dataset and dimension and stored externally for reproducibility. As indicated in Line 7 of Algorithm 5, these values are then read, from a JSON file, as input before estimation.

For each λ in this set, the algorithm computes a corresponding M-matrix $\Theta_\lambda = \text{SQUIC\_FIT}(Y, \lambda)$ (Line 8 Algorithm 5). All resulting matrices are retained for downstream analysis, enabling systematic evaluation of how the regularization parameter affects graph structure and clustering outcomes.

The output of SQUIC_FIT consists of the following elements: the estimated M-matrix Θ, the number of non-zero entries (`nnz`), the average number of non-zero entries per row (`nnz/row`) and the computational time. While the last three quantities serve primarily as statistical information for monitoring the estimation process, the M-matrix Θ is central to the subsequent analysis. For diagnostic purposes, Θ is visualized as a spy plot (Line 9 Algorithm 5), which provides a qualitative assessment of the sparsity pattern. In this representation, a user whose corresponding row (and column) in Θ contains zero off-diagonal entries is uncorrelated with all others and therefore corresponds to an isolated node in the inferred graph. If an entire contiguous block of rows and columns contains zeros in all off-diagonal positions, indicates that a subset of users are mutually disconnected, forming an isolated component. Such outcomes are undesirable, as they suggest that the learned structure is incomplete or insufficiently informative. By contrast, a well-structured matrix exhibits non-zero entries for all users, ensuring that the resulting graph is connected and every node is represented. Figure 3.1 illustrates a sparsity pattern with many disconnected users, while Figure 3.2 shows a fully connected structure.

If SQUIC_FIT fails to produce a sufficiently connected structure, this typically indicates that the chosen regularization parameter λ is too large, leading to excessive sparsity. In such cases, a smaller λ is selected, and the estimation is repeated until a satisfactory connectivity level is achieved.

This stage of the pipeline concludes with the conversion of the estimated M-matrix Θ into a graph object G (Line 10 Algorithm 5), where edges are created between users i and j whenever $\Theta_{ij} \neq 0$ and edge weights are given by $|\Theta_{ij}|$. The resulting graph is then visualized using Cosmograph Python widget [43] for exploratory inspection (Line 11 Algorithm 5). Cosmograph is a Jupyter-embeddable tool built on the *anywidget* framework [44] that implements a GPU-accelerated force-directed layout: edges act as springs, nodes experience many-body repulsion, and both force computation and rendering are executed in WebGL shaders. Running the simulation entirely on the GPU minimizes host–device transfers and enables interactive layouts for large graphs.

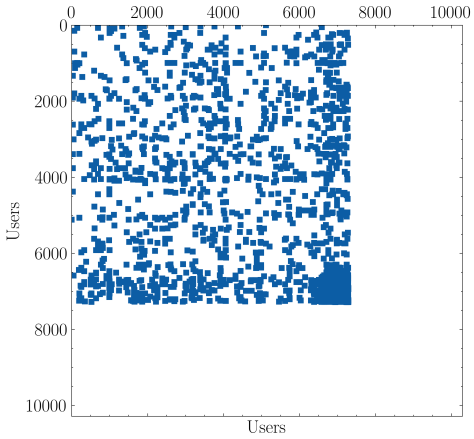

Figure 3.1. M-matrix with fragmented connectivity: nonzero entries are concentrated among a top-left subset; many users outside this subset remain disconnected.

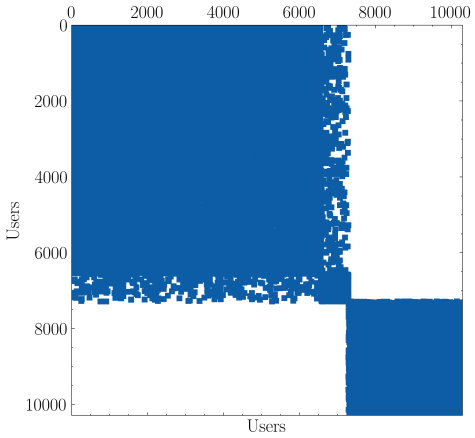

Figure 3.2. M-matrix with system-wide connectivity: nonzero entries are distributed across all users.

3.0.1.3 Community detection

The third stage of the pipeline, Line 13 Algorithm 5, is the computation of community partitions on the graph object G (or, more precisely, on the set of graphs obtained from the different precision matrices). Four clustering methods are considered, as introduced in Section 2.3.1: Spectral Clustering, Louvain, Leiden and DBSCAN. Since these methods rely on different principles, their application requires distinct inputs and parameterizations.

- Louvain (Section 2.3.1.1) and Leiden (Section 2.3.1.2). These algorithms are modularity-based community detection methods. They are applied directly to the graph object G using standard library implementations, which automatically return the partition that maximizes the modularity function (Equation 2.3).

- For DBSCAN, the input consists not only of the graph G but also of two hyperparameters: the neighborhood radius ϵ and the minimum number of points per cluster (*minPts*). Since

clustering performance is highly sensitive to these values, parameter selection is carried out through a preliminary grid search procedure for each dataset and dimension. The best-performing parameters identified in this search are then used for the final DBSCAN clustering.

- Spectral clustering was applied by implementing the standard algorithmic steps described in Algorithm 4. Starting from the graph object G, the procedure consists of constructing the (normalized) Laplacian matrix, computing its first k eigenvectors, embedding the nodes in a k-dimensional Euclidean space, and applying k-means clustering to obtain the final partition. Unlike Louvain and Leiden, spectral clustering requires the specification of the number of clusters k in advance. To determine this parameter, we adopt the eigengap heuristic [28], a widely used criterion in spectral methods. Let $\nu_1, \nu_2, \ldots, \nu_n$ denote the ordered eigenvalues of the Laplacian. The relative eigengap for dimension k is defined as:

$$\gamma_k = \frac{\nu_{k+1} - \nu_k}{\nu_k}, \quad k \geq 2. \tag{3.3}$$

  A large value of γ_k indicates that the embedding in dimension k is particularly suitable for revealing cluster structure, since it separates well-connected groups of nodes from the rest of the graph. Thus, the number of clusters k is selected as the value corresponding to the most prominent eigengap. This procedure ensures that the partitioning is not arbitrarily fixed but is instead data-driven, reflecting the intrinsic connectivity of the graph. In this way, spectral clustering complements modularity-based methods by relying on spectral properties of the Laplacian rather than optimization of a quality function.

To enable a comprehensive evaluation of community structures in the inferred transaction networks, multiple clustering algorithms are applied and compared. Louvain and Leiden optimize modularity, favoring partitions with dense intra-community connections and sparse inter-community links. DBSCAN, by contrast, is a density-based method capable of detecting arbitrarily shaped clusters and identifying outliers as noise, making it suitable when the data deviate from modularity-driven assumptions. Spectral clustering leverages the spectral properties of the Laplacian matrix, embedding the graph into a low-dimensional Euclidean space where clusters become more separable, thus offering a mathematically principled alternative. Using this diverse set of methods allows for a comparative analysis of their effectiveness on the retrieved adjacency matrices and mitigates the limitations of relying on a single clustering criterion.

In all cases, the output of each algorithm is a partition of the graph. Since the precision matrix estimation is repeated for multiple values of the regularization parameter λ, a corresponding set of partitions is obtained. These partitions are systematically stored (Line 14 Algorithm 5), forming the basis for the evaluation stage of the pipeline, where clustering quality is assessed and compared across methods and parameter settings.

3.0.1.4 Clustering evaluation

The final stage is the evaluation of the partitions obtained (Line 16 Algorithm 5) from the different regularization parameters and clustering algorithms. Since ground-truth labels are not available in the general case, evaluation is performed using internal validation metrics (see Section 2.3.4). Specifically, two complementary measures are employed, Modularity, Q, (Equation

2.3) and Partition Density, $P_{density}$, (Section 2.3.4.1). Using both metrics provides greater robustness in evaluation, as they capture distinct aspects of community quality. While modularity emphasizes separation between groups, partition density emphasizes their internal cohesiveness. The results are recorded across the set of λ values used in precision matrix estimation. The values of Q and $P_{density}$ are plotted jointly, enabling comparison of clustering quality across algorithms and regularization parameters. This visualization (Line 18 Algorithm 5) highlights parameter regimes that produce well-structured communities. Finally, Line 18 Algorithm 5, the inferred graph G are visualized with clusters highlighted in distinct colors with Cosmograph. This qualitative representation complements the quantitative metrics by providing an intuitive understanding of how clusters are distributed in the network.

Together, these steps constitute a complete processing pipeline: starting from raw transaction data, constructing user–day balance time series, estimating sparse precision matrices, deriving graph representations and extracting and evaluating community structures through multiple clustering algorithms. This common pipeline forms the methodological backbone of the study, ensuring that both datasets are analyzed in a consistent and comparable manner.

3.0.2 PaySim extension: supervised validation

Since the PaySim datasets explicitly label users as either clients (C) or merchants (M), a supervised validation experiment is enabled, the inferred graph structure is tested for its effectiveness in recovering the known two-class division. To this end, a dedicated processing pipeline is implemented (Algorithm 6).

Algorithm 6: PaySim extension Processing Pipeline

Input : Target dataset **dimension** dim (e.g., 100, 1K, 10K, 100K)

1 **Data acquisition and preprocessing**;
2 `transactions` ← Load `transactions.csv` file corresponding to dim;
3 Y ←Build users × days balance table $Y \in \mathbb{R}^{p \times n}$ from `transactions`;
4 $Y \leftarrow$ NORMALIZE(Y) ;
5 M**-matrix estimation**;
6 $\lambda \leftarrow$ Read reg. parameters associated to dataset from configuration JSON;
7 $(\Theta,$ NNZ, NNZ\ROW, TIME$) \leftarrow$ SQUIC_FIT(Y, λ);
8 **if** STRUCTURESUFFICIENT(Θ) *is* ***false*** *;* `// Visual interpretation of` Θ
9 **then**
10 $M_{\text{KNN}} \leftarrow$ KNN(Y, K);
11 $\Theta \leftarrow$ SQUIC_FIT($Y, \lambda, M_{\text{KNN}}$);
12 $G \leftarrow$ MMATRIXTOGRAPH(Θ) ; `// edge if` $\Theta_{ij} \neq 0$, `weight` $|\Theta_{ij}|$
13 COSMOGRAPHVISUALIZE(G);
14 **Community detection**;
15 PARTITIONS $\leftarrow$ COMPUTESPECTRALCLUSTERING($G, k = 2$);
16 **Clustering evaluation**;
17 (F1, ARI) $\leftarrow$ EVALUATECLUSTERING(G, PARTITIONS);
18 PLOTRESULTS(G, F1, ARI);
19 COSMOGRAPHVISUALIZE(G, PARTITIONS);
20 **End**;

3.0.2.1 Data acquisition and preprocessing

This experiment begins with the same acquisition step as in the common pipeline: the user specifies the dataset dimension (100, 1K, 10K or 100K), which determines the corresponding CSV file to load (Line 2 Algorithm 6). The transaction file is then transformed into the users–day balance matrix Y (Line 3 Algorithm 6).

In this case, however, decomposition is deliberately omitted. As discussed in Section 2.2.1, decomposition removes systematic temporal components, but in PaySim it also eliminates the pronounced differences in balance dynamics between clients and merchants (Figure 3.5). Visual inspection of the balance time series reveals systematic behavioral differences: client balances typically exhibit an overall upward trend, whereas merchant balances trend downward (Figure 3.3). Since these differences are precisely the signal we wish to capture in the supervised setting, preprocessing is limited to normalization only (Line 4 Algorithm 6), applied according to Equation 3.2. This ensures that magnitude effects are removed while preserving the shape characteristics that distinguish the two user classes (3.4).

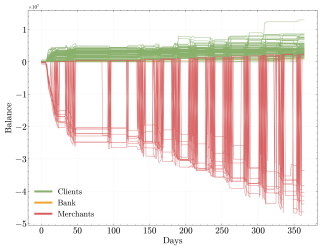

Figure 3.3. PaySim 100 Time-series

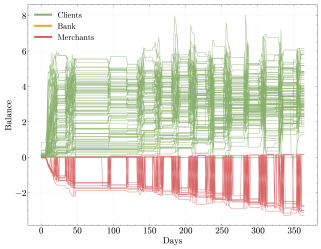

Figure 3.4. PaySim 100 Time-series after normalization

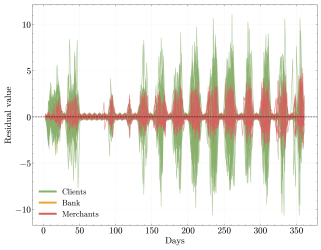

Figure 3.5. PaySim 100 Time-series after decomposition

Although the i.i.d. assumption is not satisfied in these datasets, this does not preclude the use of SQUIC-Fit algorithm. As noted in [10], in the context of Gaussian Markov Random Fields it is common to assume that the observed data matrix $Y \in \mathbb{R}^{p \times n}$ consists of n i.i.d. samples from a p-variate Gaussian distribution $\mathcal{N}(\mu, \Sigma)$, where Σ is the true covariance matrix. However, even when this assumption is not strictly met, estimating the covariance matrix Σ or its inverse (the precision matrix $\Theta = (\Sigma^*)^{-1}$) can still yield useful information for downstream tasks such as graph learning.

3.0.2.2 M-matrix estimation

As in the common pipeline, the next stage is the computation of the M-matrix using the SQUIC-Fit algorithm. A selected set of regularization parameters λ is previously defined then loaded (Line 6 Algorithm 6), and for each λ the algorithm estimates a corresponding precision matrix Θ (Line 7 Algorithm 6). All resulting matrices are retained for downstream analysis, enabling systematic evaluation of how the choice of λ influences graph structure and clustering outcomes.

Each precision matrix is then inspected through visualization of its sparsity pattern (Line 8 Algorithm 6). If SQUIC_FIT fails to produce a sufficiently connected structure, even after reducing λ, a K-Nearest Neighbors (KNN) matrix is computed from Euclidean distances between time series (Line 10 Algorithm 6) and used as a structural prior in a subsequent run of SQUIC-Fit (Line 11 Algorithm 6).

3.0.2.3 K-Nearest Neighbor (KNN)

The K-Nearest Neighbors (KNN) method is commonly known as a non-parametric classification technique that relies on local neighborhood similarity [45]. In this work, however, KNN is not used for classification but to provide a structural prior for precision matrix estimation. Specifically, a KNN matrix is constructed by connecting each user to its k nearest neighbors based on Euclidean distance between balance time series. This matrix is then supplied to SQUIC-Fit to ensure that the resulting precision matrix reflects a sufficiently connected network.

The choice of k depends on dataset size: larger k values are employed for smaller datasets to avoid excessive fragmentation, while smaller k values are used for larger datasets to prevent overly dense connectivity. Empirical tests on PaySim confirmed that incorporating the KNN prior consistently yields a connected graph in which the estimated M-matrix captures interactions for the majority of users, whereas without this bias the matrix remains to sparse and fails to represent most of the population.

This stage of the pipeline concludes with the conversion of the estimated precision matrix Θ into a graph object G (Line 12 Algorithm 6), followed by its visualization with Cosmograph (Line 13 Algorithm 6).

3.0.2.4 Clustering with spectral methods

Following precision matrix estimation and graph construction, clustering is performed using spectral clustering (Line 15 Algorithm 6). Unlike the common pipeline, which applies four different algorithms, this extension relies exclusively on spectral clustering, since it is the only method among those considered that permits the number of clusters to be specified a priori. In this experiment, the number of clusters is fixed to $k = 2$ in order to test whether the inferred graph structure can effectively separate clients from merchants. The procedure follows the algorithmic steps outlined in Algorithm 4.

As in the common pipeline, precision matrix estimation is repeated for multiple values of the regularization parameter λ. Consequently, a corresponding set of spectral partitions is obtained. These partitions are systematically stored to support the subsequent evaluation stage, where clustering quality is assessed and compared across parameter settings.

3.0.2.5 Clustering evaluation

The final stage of the pipeline is the evaluation of the partitions obtained from the different regularization parameters (Line 17 Algorithm 6). Unlike in the common pipeline, where no labels are available, the PaySim datasets provide ground-truth labels distinguishing clients and merchants. This makes it possible to evaluate clustering quality using external validation metrics, specifically, two complementary measures are employed, F1-Score (Section 2.3.3.1) and ARI (Section 2.3.3.2). The results are recorded across the set of λ values used in precision matrix estimation. The values of F1 and ARI are plotted jointly (Line 18 Algorithm 6), enabling comparison of clustering quality across algorithms and regularization parameters.

Finally, the inferred graph G are visualized with clusters highlighted in distinct colors with Cosmograph (Line 19 Algorithm 6).

Chapter 4

Numerical experiments

This chapter presents the results obtained from the experiments described in Chapter 3. A central element across all experiments is the role of the regularization parameter λ, which controls the sparsity of the precision matrix inferred by SQUIC-FIT. For each dataset size, a different subset of λ values is explored. These subsets were selected through a grid-search process aimed at identifying the parameter ranges where meaningful structures emerge, while avoiding trivially sparse or excessively dense solutions.

To maintain readability, this chapter presents only the key results and representative figures. Detailed tables and supplementary figures are provided in Appendix A.2 for the AMLSim and PaySim common experiment and in Appendix A.3 for the PaySim extension.

4.1 AMLSim and PaySim: common experiment

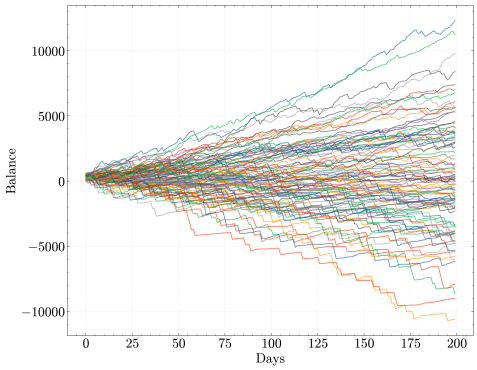

Figure 4.1. Raw AMLSim100 time series

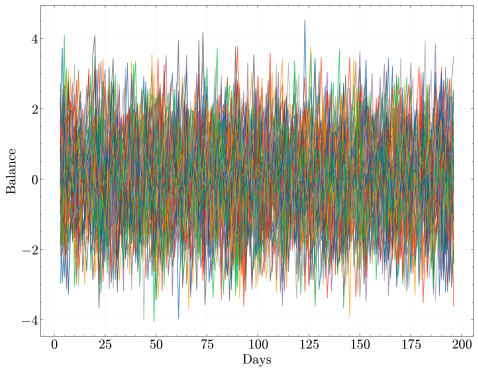

Figure 4.2. Residual of AMLSim100 after decomposition and normalization

This section reports the outcomes of applying the common experimental pipeline (Section 3.0.1) to the AMLSim and PaySim datasets across four dimensions (100, 1K, 10K and 100K users). The analysis follows the experimental pipeline described in Section 3.0.1, beginning with an evaluation of SQUIC-Fit in terms of sparsity and runtime, followed by clustering with four

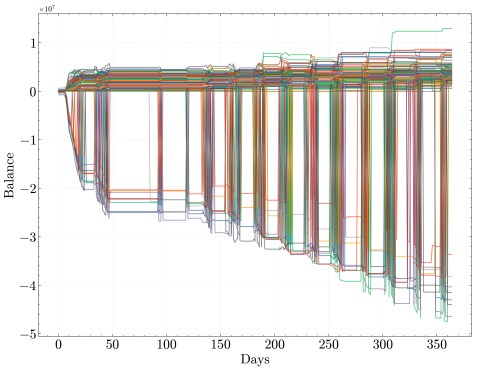

Figure 4.3. Raw PaySim100 time series

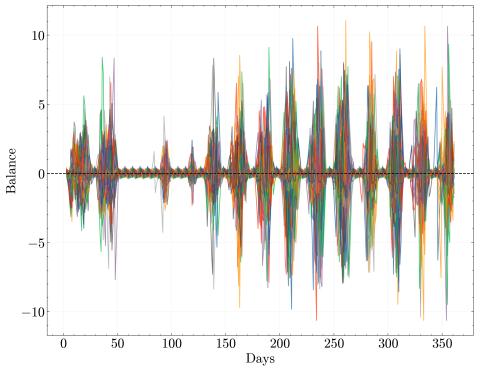

Figure 4.4. Residual of PaySim100 time series after decomposition and normalization

methods (Louvain, Leiden, Spectral and DBSCAN). Representative graph visualizations are then provided to illustrate the qualitative characteristics of the detected communities.

Before presenting the network inference and clustering results, Figures 4.1–4.4 illustrate representative examples of time series for both AMLSim and PaySim (100-user datasets), shown before and after decomposition/normalization. Other dataset dimensions exhibit similar behavior and are omitted for brevity.

4.1.0.1 SQUIC-Fit results

Tables A.1 and A.2 summarize SQUIC-Fit performance on AMLSim and PaySim respectively. The tables reports runtime, the number of non-zero entries (nnz), and the average nnz per row (nnz/row) along with the derived parameter $\eta = \lambda/10$. In both datasets, decreasing λ leads to denser precision matrices and higher computational cost, though the scaling effect is more pronounced in larger dimensions.

Figures A.5–A.10 illustrate the relationship between λ, runtime and matrix sparsity for AMLSim and PaySim across dataset sizes. In both cases, decreasing λ results in a rapid increase in nnz/row, which is accompanied by significantly higher computational costs, especially for the larger datasets (10K and 100K users). This confirms the expected trade-off between sparsity and efficiency in the SQUIC-Fit estimation process.

An illustration of the learned structures is provided in Figures 4.5–4.8, which report the precision matrices (spy plots) obtained for $\lambda = 0.2$ on the 100-user AMLSim and PaySim datasets, together with their associated graphs visualized in Cosmograph. In both datasets, the structures visible in the precision matrices correspond to regions of non-zero entries, which in the graph visualizations manifest as sets of nodes grouped with edges between each other. In these visualizations, node size reflects the node's degree, while edge thickness is proportional to the absolute weight of the corresponding entry in the precision matrix. These qualitative results complement the quantitative metrics by highlighting how the choice of λ influences the emergence of community structure in the inferred networks.

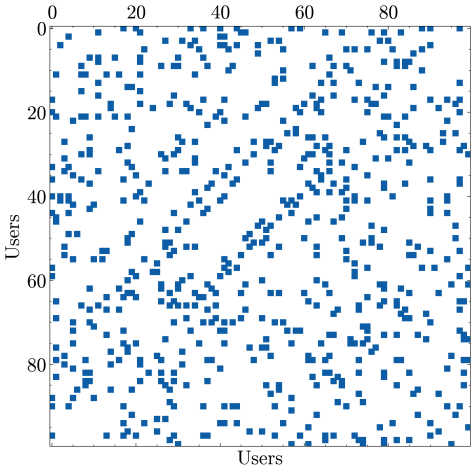

Figure 4.5. SQUIC-Fit M-matrix for AML-Sim100 with $\lambda = 0.2$

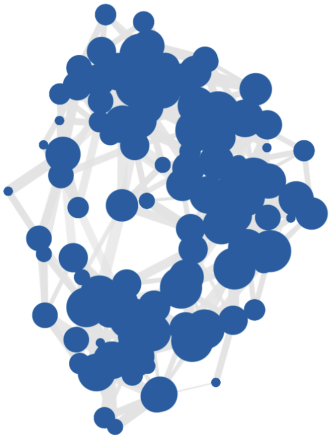

Figure 4.6. Associated graph visualized with Cosmograph (AMLSim100, $\lambda = 0.2$)

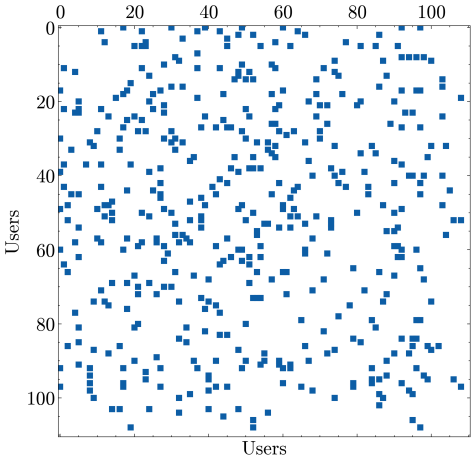

Figure 4.7. SQUIC-Fit M-matrix for PaySim100 with $\lambda = 0.2$

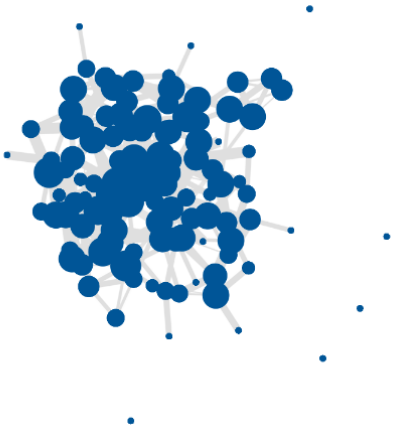

Figure 4.8. Associated graph visualized with Cosmograph (PaySim 100, $\lambda = 0.2$)

4.1.0.2 Clustering results

Clustering performance for both AMLSim and PaySim is reported in Tables A.3–A.10, which report the number of clusters, modularity and partition density across λ values and clustering methods. Louvain and Leiden consistently achieve the highest modularity, although often with a very large number of clusters, while Spectral Clustering produces fewer clusters but lower Q values. DBSCAN shows unstable behavior, sometimes yielding very fragmented partitions.

Figures 4.9–4.16 complement these results by visualizing the evolution of clustering metrics across λ, highlighting the trade-offs between sparsity, modularity, and partition density for the two datasets at each scale.

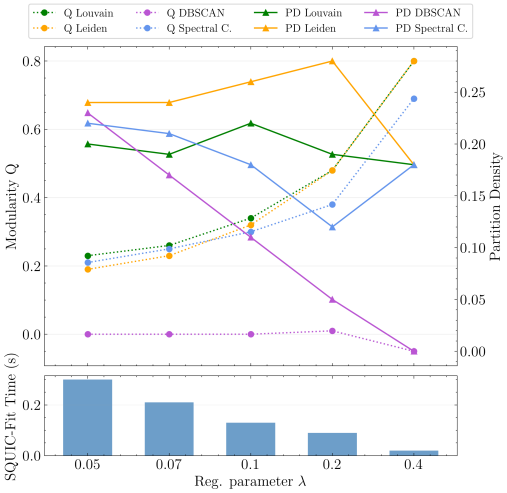

Figure 4.9. Clustering metrics on AMLSim100

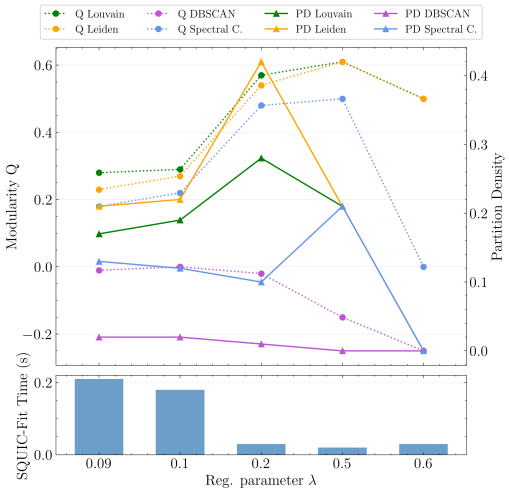

Figure 4.10. Clustering metrics on PaySim100

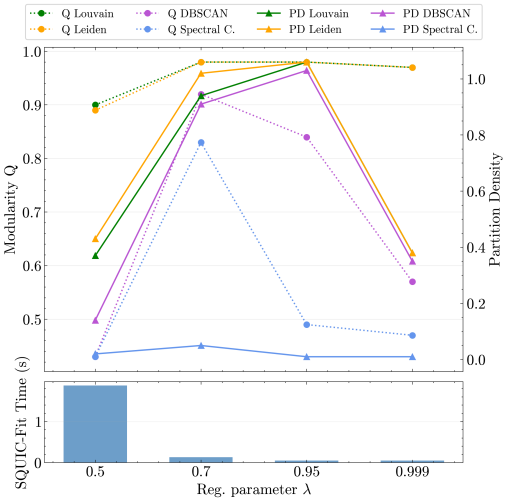

Figure 4.11. Clustering metrics on AMLSim1K

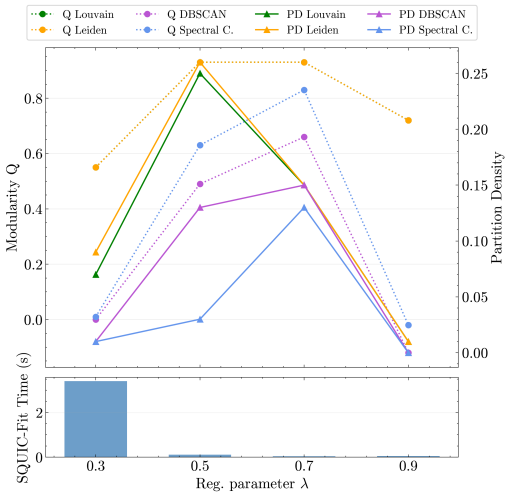

Figure 4.12. Clustering metrics on PaySim1K

The choice of the regularization parameter λ plays a critical role in determining the inferred network structure and the resulting clustering performance. When λ is set too high, the precision matrices become excessively sparse, with very few edges in the corresponding graphs. In this regime, community detection is hindered by the lack of connections, leading to partitions with limited structural significance and low values of modularity and partition density. In contrast, when λ is too low, the graphs become overly dense with nearly all nodes connected. This excessive connectivity makes it difficult to identify meaningful community boundaries, resulting in poor clustering performance.

The most informative results arise in the intermediate range of the regularization parameter, where sparsity and density are balanced. In these cases, the curves of Q and P_{density} exhibit a clear peak, indicating that the detected communities are both well-connected internally and

sufficiently separated from each other. Several of the reported plots display this desirable "bell-shaped" trend, with the center of the regularization parameter range yielding the best clustering outcomes. However, this behavior was not consistently observed across all datasets and scales, highlighting how the interaction between λ, dimensionality and intrinsic dataset characteristics can strongly affect results. In addition, computational costs at very low λ values were sometimes prohibitive, limiting exploration of denser regimes.

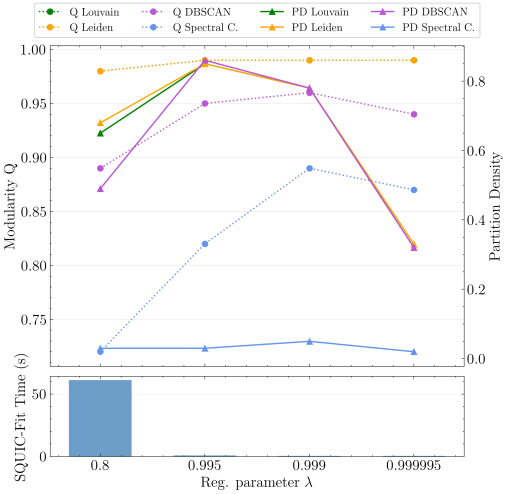

Figure 4.13. Clustering metrics on AMLSim10K

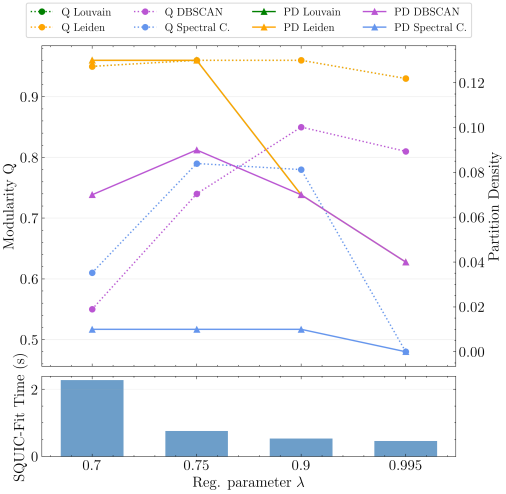

Figure 4.14. Clustering metrics on PaySim10K

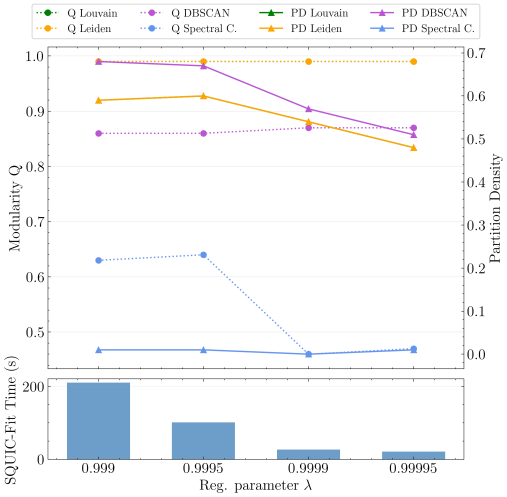

Figure 4.15. Clustering metrics on AMLSim100K

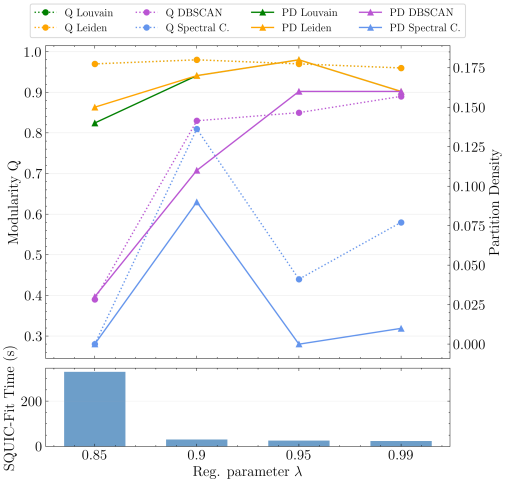

Figure 4.16. Clustering metrics on PaySim100K

Another key observation is the very large number of clusters produced by Louvain, Leiden and DBSCAN, particularly for the larger datasets. These methods do not require the number of clusters to be specified in advance, but instead aim to optimize their respective quality functions. As a result, many nodes, especially weakly connected or peripheral ones, are isolated into small

clusters, inflating the total cluster count. Spectral Clustering, by contrast, relies on the eigengap heuristic [28] to estimate the number of clusters, which typically yields far fewer partitions. While this avoids fragmentation, it penalizes Spectral's performance in terms of Q and P_{density}, as the coarse partitions often merge nodes that would form distinct communities under other methods.

As outlined in Algorithm 5, the last step of the pipeline visualizes the cluster assignments on the inferred graphs. Figures 4.17–4.24 compare, for $\lambda = 0.2$ in the 100-user setting, the partitions obtained by the four clustering methods, with AMLSim and PaySim shown side by side. Across both datasets, Louvain and Leiden consistently identify fine-grained partitions, yielding a large number of small communities that cover nearly all nodes. By contrast, Spectral clustering produces much coarser structures, typically grouping the graph into only a few large communities. DBSCAN, instead, generates fragmented partitions: many points are isolated or placed into very small clusters, resulting in scattered and less interpretable structures.

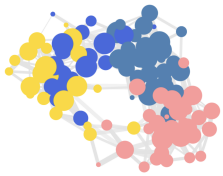

Figure 4.17. Louvain clustering on AMLSim100 ($\lambda = 0.2$), 4 clusters

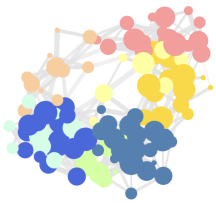

Figure 4.18. Leiden clustering on AMLSim100 ($\lambda = 0.2$), 7 clusters

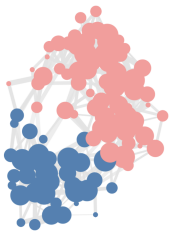

Figure 4.19. Spectral clustering on AMLSim100 ($\lambda = 0.2$), 2 clusters

Figure 4.20. DBSCAN clustering on AMLSim100 ($\lambda = 0.2$), 17 clusters

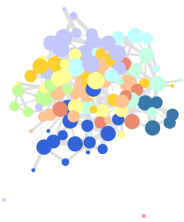

Figure 4.21. Louvain clustering on PaySim100 ($\lambda = 0.2$), 15 clusters

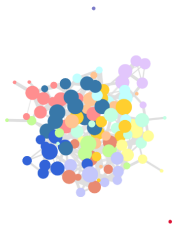

Figure 4.22. Leiden clustering on PaySim100 ($\lambda = 0.2$), 18 clusters

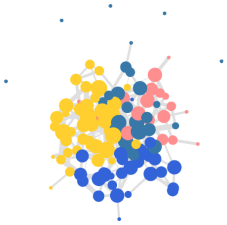

Figure 4.23. Spectral clustering on PaySim100 ($\lambda = 0.2$), 4 clusters

Figure 4.24. DBSCAN clustering on PaySim100 ($\lambda = 0.2$), 82 clusters

4.2 PaySim extension: supervised validation

The purpose of this chapter is to evaluate the performance of SQUIC-Fit combined with Spectral Clustering on the PaySim dataset at different scales: 100, 1K, 10K and 100K users. Because PaySim explicitly labels each user as either a client (C) or a merchant (M), the inferred graph structures can be directly validated against this ground truth (Section 3.0.2).

As discussed in the general introduction to this chapter, the results depend critically on the choice of the regularization parameter λ. Here, this parameter is selected specifically on how SQUIC-Fit combined with Spectral Clustering performs on PaySim datasets of increasing size, with and without the inclusion of a KNN bias.

Another key aspect is the inclusion of a KNN bias matrix (Line 10/11 of Algorithm 6). This bias, constructed from the time series of user balances, enforces local connectivity that helps guide the estimation process. The effect of the bias varies with dataset size: for PaySim100, the natural block structure between clients and merchants is already captured by SQUIC-Fit, and no bias is needed. For PaySim1K, good clustering can still be achieved without bias, though only at the cost of long runtimes and structurally imbalanced graphs. With bias, comparable clustering quality is obtained in significantly shorter time. While for PaySim10K and PaySim100K, meaningful results cannot be reached without bias, as the computational cost required to obtain dense matrices becomes prohibitive. In these cases, the bias is essential to make the problem tractable and to recover the client–merchant structure.

The results are presented separately for each dataset size, highlighting both the role of λ and the contribution of the bias.

4.2.1 PaySim100

Table 4.1 reports the statistics of the precision matrices estimated with SQUIC-Fit for different values of the regularization parameter λ. For each configuration, the runtime, number of nonzero entries (*nnz*) and average nnz per row of the resulting matrix (*nnz/row*) are provided. As expected, higher values of the regularization parameter yield sparser matrices, whereas lower values produce denser structures with more edges per node.

Table 4.1. SQUIC-Fit results on PaySim100

λ	Time (s)	nnz	nnz/row
0.6	0.09	1414	12.74
0.5	0.13	1662	14.97
0.4	0.28	1704	15.35
0.3	0.29	1666	15.01
0.2	0.65	1604	14.45
0.1	1.69	1820	16.40

Figures 4.25 and 4.26 show the precision matrix obtained with $\lambda = 0.3$ and the correlated graph visualized with Cosmograph. In the precision matrix (left) is it possible to notice that some rows are entirely empty of non-zero values, meaning that for certain users no conditional dependencies are detected. In the associated graph (right), this translates into isolated nodes.

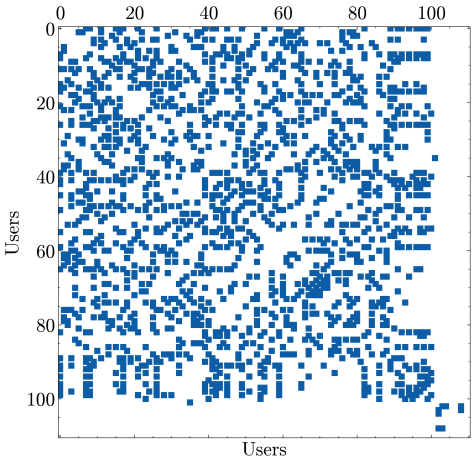

Figure 4.25. SQUIC-Fit precision matrix for $\lambda = 0.3$ (PaySim100)

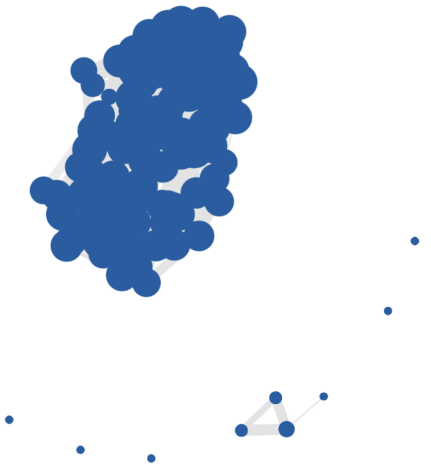

Figure 4.26. Associated graph visualized with Cosmograph ($\lambda = 0.3$)

The trade-off between sparsity and density influences the performance of subsequent clustering. Table 4.2 reports the clustering performance of Spectral Clustering applied to the graphs inferred by SQUIC-Fit, where the number of clusters (fixed to two), ARI and F1 scores are shown for each λ.

Table 4.2. Spectral Clustering results on PaySim100

λ	n_{clusters}	ARI	F1
0.6	2	-0.03	0.58
0.5	2	1.00	1.00
0.4	2	1.00	1.00
0.3	2	0.93	0.99
0.2	2	0.80	0.97
0.1	2	-0.06	0.51

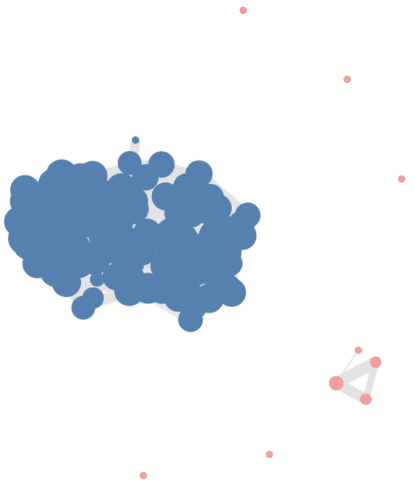

Figure 4.27. Spectral Clustering on PaySim100 ($\lambda = 0.3$)

As showed also in Figure 4.28, the clustering results are excellent for intermediate values of λ (0.3–0.5), where both ARI and F1 reach values close to or equal to 1.0, indicating perfect recovery of the client–merchant partition; a visual interpretation of this result is showed in Figure 4.27. Instead, at extreme values of λ the performance degrades, for $\lambda = 0.6$, the graph becomes overly sparse, producing near-random assignments, while for $\lambda = 0.1$, the excessive

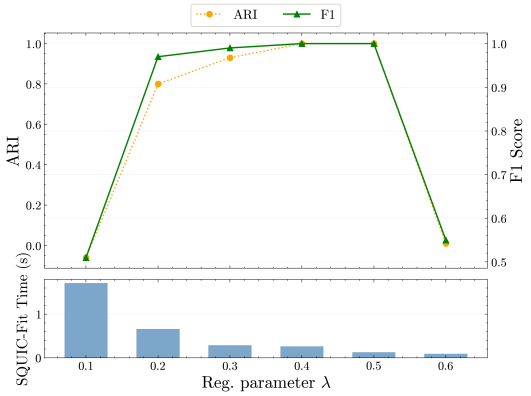

Figure 4.28. Spectral Clustering metrics on PaySim100

density blurs the community structure.

Overall, the Spectral Clustering pipeline achieves near-perfect segmentation on PaySim-100 without requiring any additional bias. Although some merchants appear as isolated nodes in the inferred graph, this is consistent with the underlying ground truth: isolated nodes are correctly identified as merchants. The natural block structure between clients and merchants is thus already captured by SQUIC-Fit, provided that λ is chosen within a reasonable range. Introducing a KNN bias is therefore unnecessary at this scale.

4.2.2 PaySim1K

At the 1K scale, SQUIC-Fit without bias can still achieve reasonable clustering performance, but only at low values of λ. In this regime, the precision matrices become extremely dense, strongly connecting a block of users while leaving others isolated. Spectral Clustering then trivially assigns all connected users to one cluster and the isolated nodes to another.

The resulting ARI and F1 scores are acceptable (ARI ≈ 0.8, F1 ≈ 0.95) because this separation reflects the ground truth; however, runtimes exceed 300 seconds for $\lambda \leq 0.75$, making the approach computationally impractical (Table 4.3). The resulting statistics of SQUIC-Fit are reported in Table 4.3, and an example of the inferred precision matrix is shown in Figure 4.29. It is noticeable that the method predominantly recovers a block of users (corresponding to clients), while others (merchants) remain largely disconnected. Introducing a KNN bias with $k = 8$ addresses these limitations. Although the clustering metrics (ARI and F1) remain similar to the best no-bias results, the biased approach achieves them in significantly shorter runtimes while producing more balanced precision matrices in which both clients and merchants are represented (Table 4.4, Figure 4.30).

Table 4.3. SQUIC-Fit results on PaySim1K without bias

λ	Time (s)	nnz	nnz/row
0.90	1.44	10394	10.13
0.85	19.98	16968	16.54
0.80	44.62	19888	19.38
0.75	257.35	17954	17.50
0.70	562.91	21214	20.68

Table 4.4. SQUIC-Fit results on PaySim1K with KNN bias

λ	Time (s)	nnz	nnz/row
0.97	0.24	5264	5.13
0.95	0.28	5338	5.20
0.90	1.21	5872	5.72
0.80	10.46	10262	10.00
0.75	72.43	15764	15.36

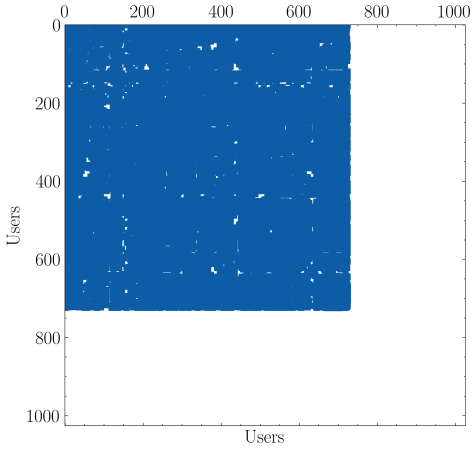

Figure 4.29. SQUIC-Fit precision matrix for $\lambda = 0.8$ (PaySim1K)

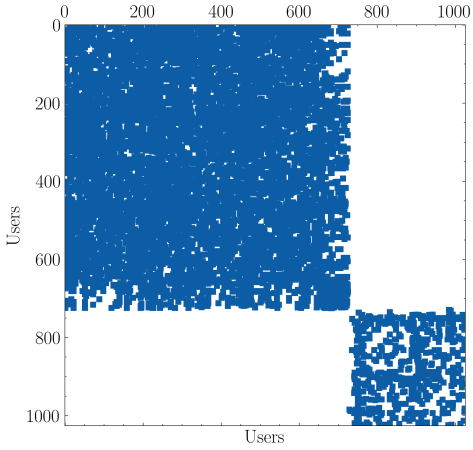

Figure 4.30. SQUIC-Fit precision matrix for $\lambda = 0.8$ with bias (PaySim1K)

Spectral Clustering now achieves ARI up to 0.97 and F1 = 0.99 (for $\lambda = 0.95$). The detailed results are reported in Table 4.5, accompanied by a visualization of ARI and F1 trends in Figure 4.31.

The introduction of the KNN bias enables effective segmentation and meaningful results in faster times. This demonstrates that from the 1K scale onwards, bias becomes essential for uncovering meaningful structure in the PaySim data.

Table 4.5. Spectral Clustering results on PaySim1K with KNN bias

λ	n_{clusters}	ARI	F1
0.97	2	0.48	0.85
0.95	2	0.97	0.99
0.90	2	0.48	0.85
0.80	2	0.48	0.85
0.75	2	0.04	0.62

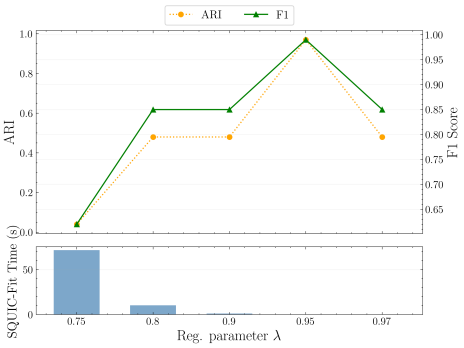

Figure 4.31. ARI vs. F1 for Spectral Clustering on PaySim1K with bias

4.2.3 PaySim10K

At the 10K scale, the situation changes. Without bias, SQUIC-Fit is unable to reach the regime where dense matrices connect the entire client block. Achieving such density would require prohibitively high runtimes, far beyond practical limits. As a result, the no-bias precision matrices remain too sparse and clustering performance stays near random (ARI ≈ 0, F1 ≈ 0.5). With the introduction of KNN bias ($k = 4$), however, the algorithm immediately produces well-structured matrices where both client and merchant communities are represented.

Table 4.6. SQUIC-Fit results on PaySim10K without bias

λ	Time (s)	nnz	nnz/row
0.999	0.45	330	0.03
0.990	0.72	2536	0.25
0.980	3595.69	31410	3.05

Table 4.7. SQUIC-Fit results on PaySim10K with KNN bias

λ	Time (s)	nnz	nnz/row
0.9995	3.20	34492	3.35
0.9992	3.13	34496	3.35
0.9990	3.14	34490	3.35
0.9970	3.11	34484	3.35
0.9950	3.14	34498	3.35

Tables 4.6 and 4.7 reports the statistics of the estimated precision matrices before and after the introduction of the bias. The number of nonzeros indicate that the bias successfully enforces structure across the graph, as also showed by Figures 4.32 and 4.33.

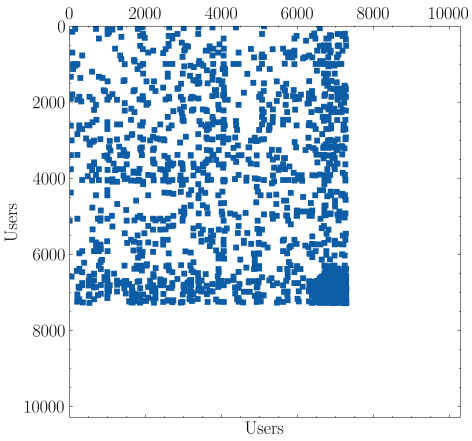

Figure 4.32. Example of SQUIC-Fit precision matrix for $\lambda = 0.99$ (PaySim10K)

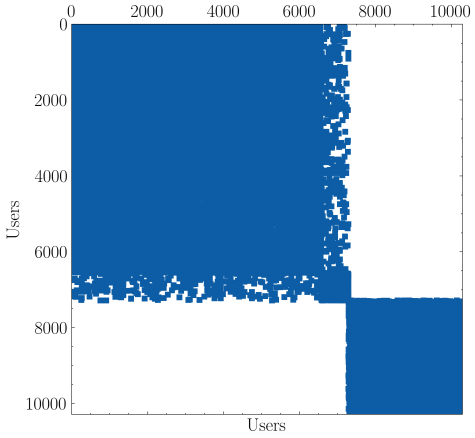

Figure 4.33. Example of SQUIC-Fit precision matrix with KNN bias for $\lambda = 0.999$ (PaySim10K)

The clustering results with the introduction of the bias improve dramatically, with ARI reaching up to 0.94 and F1 up to 0.99. Table 4.8 summarizes the results and Figure 4.9 shows the ARI–F1 trade-off across λ values.

Table 4.8. Spectral Clustering results on PaySim10K with KNN bias

λ	n_{clusters}	ARI	F1
0.9995	2	0.49	0.86
0.9992	2	0.90	0.97
0.9990	2	0.94	0.99
0.9970	2	0.93	0.98
0.9950	2	0.94	0.98

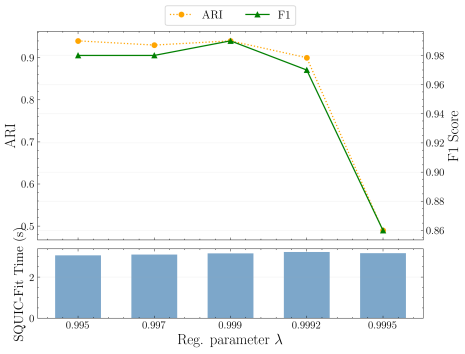

Table 4.9. ARI vs. F1 for Spectral Clustering on PaySim10K with KNN bias

4.2.4 PaySim100K

At the 100K scale, the same limitation becomes even more severe. Without bias, SQUIC-Fit would require low λ values to obtain dense matrices, but the computational cost grows prohibitively, making such experiments infeasible. Consequently, no meaningful clustering can be obtained without bias at this scale. With KNN bias ($k = 4$), SQUIC-Fit produces stable graph structures despite still requiring high runtimes.

Tables 4.10 and 4.11 reports the statistics of the estimated precision matrices. Although the runtime increases significantly (over 350 seconds), the number of non zero elements suggests that the bias maintains a consistent graph structure across λs. An example precision matrix is shown in Figure 4.35.

Table 4.10. SQUIC-Fit results on PaySim100K without bias

λ	Time (s)	nnz	nnz/row
0.996	25.29	47548	0.47
0.995	26.59	55068	0.54
0.994	61.46	67626	0.66

Table 4.11. SQUIC-Fit results on PaySim100K with KNN bias

λ	Time (s)	nnz	nnz/row
0.99999	361.48	268964	2.63
0.99992	359.66	268734	2.63
0.99991	358.44	268728	2.63
0.99990	367.78	268706	2.63

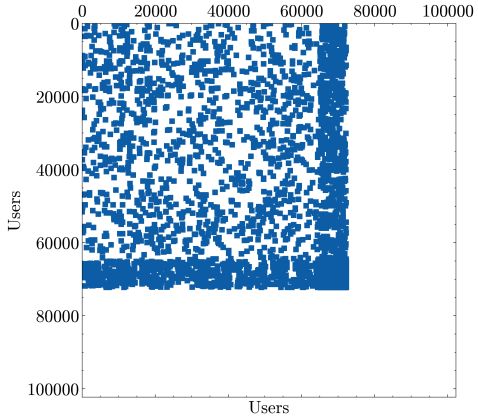

Figure 4.34. Example of SQUIC-Fit precision matrix for $\lambda = 0.995$ (PaySim100K)

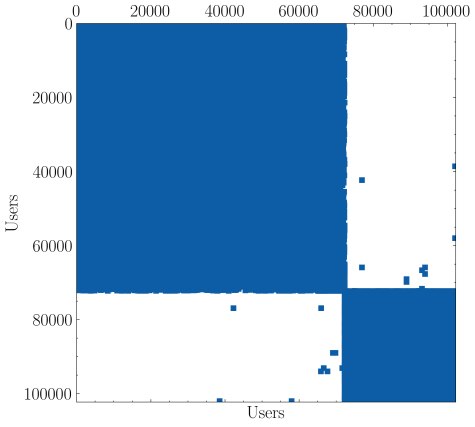

Figure 4.35. Example of SQUIC-Fit precision matrix with KNN bias for $\lambda = 0.99992$ (PaySim100K)

Clustering results confirm that the bias remains effective, although sensitivity to λ increases at this scale. The best performance is achieved at $\lambda = 0.99992$ with ARI $= 0.68$ and F1 $= 0.92$. Results are summarized in Table 4.12, with trends visualized in Figure 4.13.

Table 4.12. Spectral Clustering results on PaySim100K with KNN bias

λ	n_{clusters}	ARI	F1
0.99999	2	-0.00	0.60
0.99992	2	0.68	0.92
0.99991	2	0.44	0.83
0.99990	2	0.14	0.70

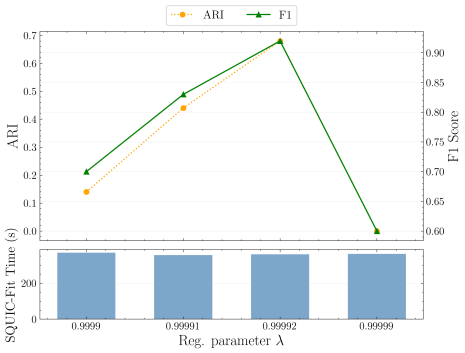

Table 4.13. ARI vs. F1 for Spectral Clustering on PaySim100K with KNN bias

While the accuracy is lower than at smaller scales, these results demonstrate that the bias is

essential for making the problem tractable at 100K users.

4.2.5 Conclusions

Across the four dataset sizes, a clear pattern emerges (Figure 4.36). At small scale (PaySim100), the natural client–merchant division is perfectly recovered without bias. At 1K users, good results are possible without bias but only at the cost of impractical runtimes; the bias provides efficiency and structural balance. At 10K and 100K, meaningful results cannot be obtained without bias, which becomes essential both for tractability and for uncovering the client–merchant structure. Thus, the KNN bias evolves from being unnecessary (100), to advantageous (1K), to indispensable (10K, 100K).

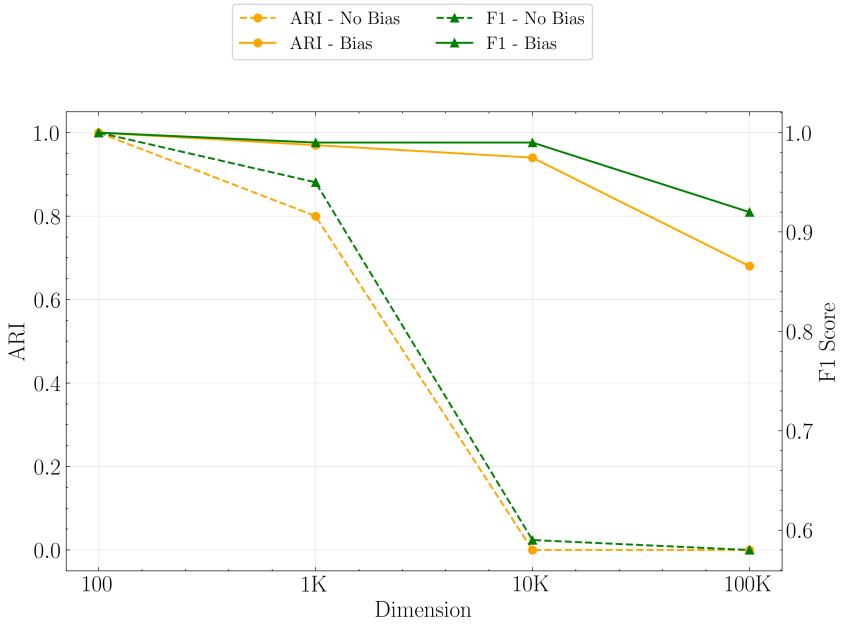

Figure 4.36. External metrics comparison between PaySim datasets

Chapter 5

Conclusion

This thesis has investigated the problem of client segmentation in financial services through unsupervised graph-based clustering applied to account balance time series. Motivated by the limitations of traditional static segmentation techniques, the study developed and evaluated a novel methodological framework that infers behavioral similarity from aggregate account-level data rather than explicit transaction records. The proposed pipeline combines the *Sparse QUadratic approximation for Inverse Covariance* (SQUIC) algorithm with multiple clustering approaches, Louvain, Leiden, Spectral Clustering and DBSCAN.

Two synthetic datasets, PaySim and AMLSim, were used to validate the approach. The framework was applied first in an unsupervised setting, where clusters were assessed via internal quality metrics such as modularity and partition density, and subsequently in a supervised validation experiment on PaySim, using the ground-truth client–merchant labels to test external validity.

The numerical experiments highlight several important findings. First, the regularization parameter λ in the SQUIC algorithm proved crucial in shaping the inferred network structures and downstream clustering performance. A balanced choice of λ led to networks that captured meaningful community structures, as indicated by peaks in modularity and partition density, whereas extreme values produced either overly sparse or overly dense graphs with limited interpretability. Second, the comparison of clustering algorithms revealed trade-offs between granularity and stability. Louvain, Leiden and DBSCAN frequently generated a high number of small communities, reflecting fine-grained structural differences but complicating interpretability. Spectral Clustering, in contrast, produced coarser partitions with fewer clusters, sometimes at the expense of community quality. Third, the supervised study on PaySim datasets underscored the importance of incorporating a KNN-based bias into the graph construction process. While unnecessary at the smallest scale, the bias became progressively advantageous and eventually indispensable when the dataset size increased, both in terms of computational feasibility and in uncovering the meaningful client–merchant structure. This finding demonstrates that methodological adaptations are required to maintain segmentation quality in larger-scale settings.

Alongside these contributions, several limitations must be acknowledged. First, the reliance on synthetic datasets restricts the direct generalizability of results to real-world banking environments, where data distributions and noise characteristics may differ. Second, the evaluation primarily relied on internal clustering metrics, which, although informative, cannot fully

capture the practical relevance of discovered segments for business applications. Third, the computational demands of the SQUIC algorithm imposed practical constraints. At lower regularization levels, the precision matrices become increasingly dense, leading to high memory requirements and long fitting times. This made it difficult to systematically explore very low values of λ, thereby limiting the analysis of dense graph regimes (see Appendix A.1 for hardware specifications). Fourth, the scope of the study was restricted to time series derived from transaction data. While this abstraction was intentional, simulating contexts with limited visibility, it also meant that richer behavioral or contextual attributes (e.g., client demographics, transaction categories or known risk indicators) were not incorporated. As a result, the interpretability of clusters was confined to structural patterns of similarity, without the possibility of linking them to broader business or risk-related insights.

Future research could extend this work in several directions. First, applying the framework to real-world financial datasets would provide stronger evidence of its practical utility. Second, incorporating client-level attributes alongside behavioral patterns would enable a deeper analysis of the discovered segments. For example, correlating cluster structures with risk classifications could reveal whether certain communities are associated with elevated financial crime risks. Establishing such connections would provide direct evidence of the framework's applicability to compliance monitoring and risk management, thereby increasing its strategic value.

To conclude, this thesis demonstrates that time-series-based graph learning combined with clustering provides a viable and interpretable approach to client segmentation under limited data visibility. By capturing structural behavioral patterns without requiring detailed transaction records, the proposed framework contributes a flexible tool for advancing adaptive and behavior-aware financial analytics.

Appendix A

Supplementary material

A.1 Experimental Environment

To ensure reproducibility, this section documents the computational environment used for all experiments. The experiments were carried out on a personal laptop running macOS Sequoia 15.6.1, equipped with an Apple M2 Pro processor and 16 GB of RAM. The software environment was based on Python 3.10, with the key library dependencies and their versions listed below:

- `cosmograph==0.0.47`
- `networkx==3.4.2`
- `numpy==2.2.4`
- `pandas==2.2.3`
- `scikit-learn==1.6.1`
- `squic==1.0.4`
- `igraph==0.11.8`
- `leidenalg==0.10.2`
- `statsmodels==0.14.4`

A.2 AMLSim and PaySim common experiment additional results

The main discussion of these results is provided in Section 4.1. The tables and figures are included here to preserve the readability and flow of the main text.

Table A.1. Summary of SQUIC-Fit results on AMLSim across dataset dimensions

Dimension	λ	η	Time (s)	nnz	nnz/row
100	0.40	0.040	0.02	32	0.32
100	0.20	0.020	0.09	664	6.64
100	0.10	0.010	0.13	1304	13.04
100	0.07	0.007	0.20	1760	17.60
100	0.05	0.005	0.30	2194	21.94
1K	0.999	0.0999	0.04	202	0.20
1K	0.95	0.0950	0.05	1232	1.23
1K	0.70	0.0700	0.12	4150	4.15
1K	0.50	0.0500	1.87	6874	6.87
10K	0.999995	0.0999995	0.42	4888	0.48
10K	0.999	0.0999	0.55	39324	3.93
10K	0.995	0.0995	0.90	66674	6.66
10K	0.80	0.0800	61.12	278170	27.81
100K	0.99995	0.099995	21.01	300698	3.00
100K	0.9999	0.09999	26.50	456242	4.56
100K	0.9995	0.09995	101.29	1151454	11.51
100K	0.999	0.0999	210.37	1690536	16.90

Table A.2. Summary of SQUIC-Fit results on PaySim across dataset dimensions

Dimension	λ	η	Time (s)	nnz	nnz/row
100	0.60	0.060	0.03	4	0.04
100	0.50	0.050	0.02	10	0.09
100	0.20	0.020	0.03	468	4.22
100	0.10	0.010	0.18	1518	13.68
100	0.09	0.009	0.21	1702	15.33
1K	0.90	0.090	0.06	10	0.01
1K	0.70	0.070	0.04	126	0.12
1K	0.50	0.050	0.11	1046	1.02
1K	0.30	0.030	3.42	6986	6.81
10K	0.995	0.0995	0.46	280	0.03
10K	0.90	0.090	0.53	874	0.08
10K	0.75	0.075	0.76	5372	0.52
10K	0.70	0.070	2.27	9422	0.92
100K	0.99	0.099	23.27	35204	0.34
100K	0.95	0.095	25.42	54056	0.53
100K	0.90	0.090	30.22	90296	0.88
100K	0.85	0.085	329.60	146738	1.44

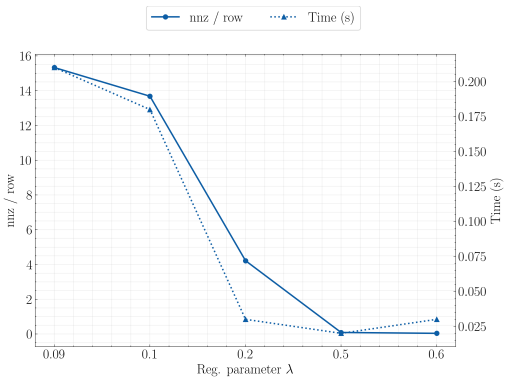

Figure A.1. Runtime vs. nnz/row on AMLSim100

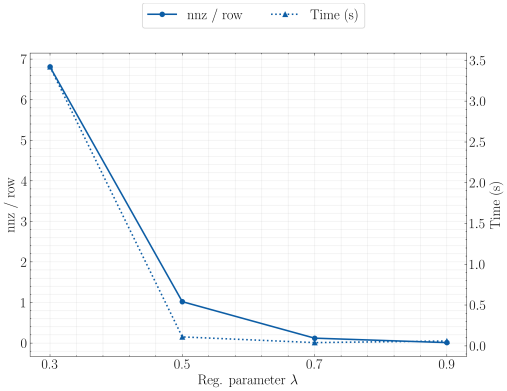

Figure A.2. Runtime vs. nnz/row on AMLSim1K

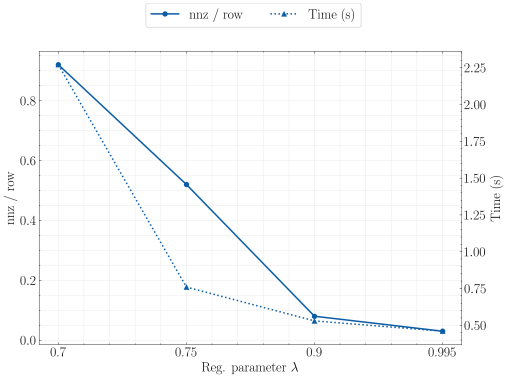

Figure A.3. Runtime vs. nnz/row on AMLSim10K

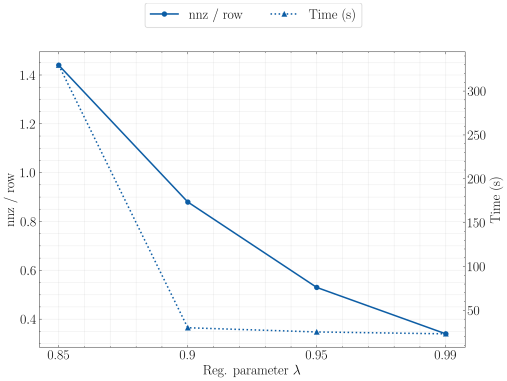

Figure A.4. Runtime vs. nnz/row on AMLSim100K

Figure A.5. Comparison of runtime and matrix density across AMLSim dataset sizes

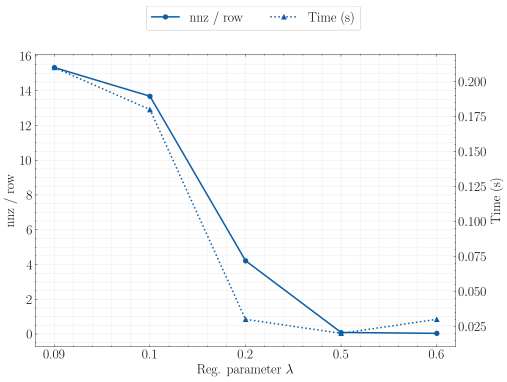

Figure A.6. Runtime vs. nnz/row on PaySim100

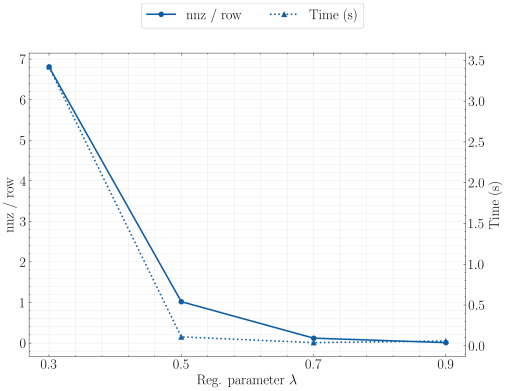

Figure A.7. Runtime vs. nnz/row on PaySim1K

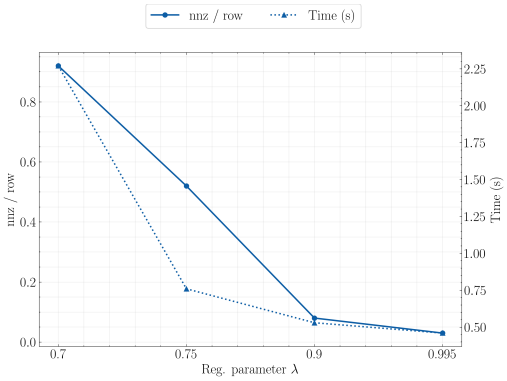

Figure A.8. Runtime vs. nnz/row on PaySim10K

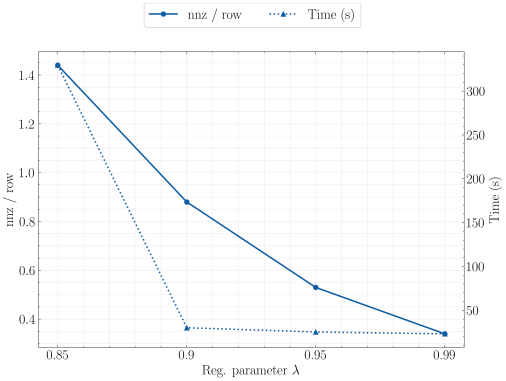

Figure A.9. Runtime vs. nnz/row on PaySim100K

Figure A.10. Comparison of runtime and matrix density across PaySim dataset sizes

Table A.3. Clustering results on AMLSim100 across λ values

λ	Method	#Clusters	P_{density}	Q
0.40	Louvain	84	0.18	0.80
	Leiden	84	0.18	0.80
	DBSCAN	100	0.00	-0.05
	Spectral	5	0.18	0.69
0.20	Louvain	5	0.19	0.48
	Leiden	7	0.28	0.48
	DBSCAN	19	0.05	0.01
	Spectral	2	0.12	0.38
0.10	Louvain	5	0.22	0.34
	Leiden	6	0.26	0.32
	DBSCAN	15	0.11	0.00
	Spectral	2	0.18	0.30
0.07	Louvain	6	0.19	0.26
	Leiden	6	0.24	0.23
	DBSCAN	6	0.17	0.00
	Spectral	2	0.21	0.25
0.05	Louvain	5	0.20	0.23
	Leiden	7	0.24	0.18
	DBSCAN	1	0.23	0.00
	Spectral	2	0.22	0.21

Table A.4. Clustering results on PaySim100 across λ values

λ	Method	#Clusters	P_{density}	Q
0.60	Louvain	109	0.00	0.50
	Leiden	109	0.00	0.50
	DBSCAN	111	0.00	-0.25
	Spectral	2	0.00	0.00
0.50	Louvain	107	0.21	0.61
	Leiden	107	0.21	0.61
	DBSCAN	111	0.00	-0.15
	Spectral	2	0.21	0.50
0.20	Louvain	15	0.28	0.57
	Leiden	18	0.42	0.54
	DBSCAN	82	0.01	-0.02
	Spectral	4	0.10	0.48
0.10	Louvain	9	0.19	0.29
	Leiden	8	0.22	0.27
	DBSCAN	88	0.02	0.00
	Spectral	3	0.12	0.22
0.09	Louvain	8	0.17	0.28
	Leiden	8	0.21	0.23
	DBSCAN	78	0.02	-0.01
	Spectral	3	0.13	0.18

Table A.5. Clustering results on AMLSim1K across λ values

λ	Method	#Clusters	P_{density}	Q
0.999	Louvain	930	0.38	0.97
	Leiden	930	0.38	0.97
	DBSCAN	961	0.35	0.57
	Spectral	2	0.01	0.47
0.95	Louvain	722	1.06	0.98
	Leiden	722	1.06	0.98
	DBSCAN	768	1.03	0.84
	Spectral	2	0.01	0.49
0.70	Louvain	383	0.94	0.98
	Leiden	386	1.02	0.98
	DBSCAN	433	0.91	0.92
	Spectral	7	0.05	0.83
0.50	Louvain	144	0.38	0.89
	Leiden	146	0.42	0.89
	DBSCAN	150	0.14	0.43
	Spectral	6	0.02	0.43

Table A.6. Clustering results on PaySim1K across λ values

λ	Method	#Clusters	P_{density}	Q
0.90	Louvain	1021	0.01	0.72
	Leiden	1021	0.01	0.72
	DBSCAN	1026	0.00	-0.12
	Spectral	2	0.00	-0.02
0.70	Louvain	982	0.15	0.93
	Leiden	982	0.15	0.93
	DBSCAN	995	0.15	0.66
	Spectral	9	0.13	0.83
0.50	Louvain	726	0.25	0.93
	Leiden	726	0.26	0.92
	DBSCAN	763	0.13	0.49
	Spectral	5	0.03	0.63
0.30	Louvain	134	0.07	0.55
	Leiden	137	0.08	0.54
	DBSCAN	190	0.01	0.00
	Spectral	2	0.01	0.01

Table A.7. Clustering results on AMLSim10K across λ values

λ	Method	#Clusters	P_{density}	Q
0.999995	Louvain	9330	0.33	0.99
	Leiden	9330	0.33	0.99
	DBSCAN	9376	0.32	0.94
	Spectral	8	0.02	0.87
0.999	Louvain	7726	0.78	0.99
	Leiden	7726	0.78	0.99
	DBSCAN	7827	0.78	0.96
	Spectral	10	0.05	0.89
0.995	Louvain	6936	0.85	0.99
	Leiden	6936	0.85	0.99
	DBSCAN	7111	0.86	0.95
	Spectral	6	0.03	0.82
0.80	Louvain	2842	0.65	0.98
	Leiden	2847	0.68	0.98
	DBSCAN	3269	0.49	0.89
	Spectral	7	0.03	0.72

Table A.8. Clustering results on PaySim10K across λ values

λ	Method	#Clusters	P_{density}	Q
0.995	Louvain	10225	0.04	0.93
	Leiden	10225	0.04	0.93
	DBSCAN	10234	0.04	0.81
	Spectral	2	0.00	0.49
0.90	Louvain	10111	0.07	0.96
	Leiden	10111	0.07	0.96
	DBSCAN	10127	0.07	0.85
	Spectral	9	0.01	0.87
0.75	Louvain	9366	0.13	0.96
	Leiden	9362	0.13	0.96
	DBSCAN	9484	0.09	0.74
	Spectral	5	0.01	0.74
0.70	Louvain	8824	0.13	0.96
	Leiden	8821	0.13	0.95
	DBSCAN	8963	0.07	0.55
	Spectral	7	0.01	0.61

Table A.9. Clustering results on AMLSim100K across λ values

λ	Method	#Clusters	P_{density}	Q
0.99995	Louvain	91517	0.48	0.99
	Leiden	91517	0.48	0.99
	DBSCAN	92919	0.51	0.87
	Spectral	2	0.00	0.47
0.9999	Louvain	89032	0.54	0.99
	Leiden	89032	0.54	0.99
	DBSCAN	90841	0.57	0.87
	Spectral	2	0.00	0.46
0.9995	Louvain	80930	0.60	0.99
	Leiden	80930	0.60	0.99
	DBSCAN	83659	0.67	0.86
	Spectral	3	0.01	0.64
0.999	Louvain	76303	0.59	0.99
	Leiden	76303	0.59	0.99
	DBSCAN	79424	0.68	0.86
	Spectral	3	0.01	0.63

Table A.10. Clustering results on PaySim100K across λ values

λ	Method	#Clusters	P_{density}	Q
0.99	Louvain	101193	0.16	0.96
	Leiden	101193	0.16	0.96
	DBSCAN	101307	0.16	0.89
	Spectral	3	0.01	0.58
0.95	Louvain	100348	0.18	0.97
	Leiden	100348	0.18	0.97
	DBSCAN	100613	0.16	0.85
	Spectral	2	0.00	0.46
0.90	Louvain	98534	0.17	0.98
	Leiden	98534	0.17	0.98
	DBSCAN	98990	0.11	0.83
	Spectral	11	0.09	0.81
0.85	Louvain	95229	0.14	0.97
	Leiden	95226	0.15	0.97
	DBSCAN	96021	0.03	0.39
	Spectral	2	0.00	0.28

Bibliography

[1] M. Pejic Bach, S. Juković, K. Dumičić, and N. Sarlija, "Business Client Segmentation in Banking Using Self-Organizing Maps," *South East European Journal of Economics and Business*, vol. 8, 11 2014. [Online]. Available: https://doi.org/10.2478/jeb-2013-0007

[2] E. Kontautaitė, "Transaction monitoring rules: Navigating the future," 2024, last Accessed: 2025-08-07. [Online]. Available: https://amlyze.com/transaction-monitoring-rules/

[3] instnt, "Pitfalls of current transaction monitoring systems," 2022, last Accessed: 2025-08-26. [Online]. Available: https://instnt.ai/pitfalls-of-current-transaction-monitoring-systems-instnt/

[4] W. R. Smith, "Product Differentiation and Market Segmentation as Alternative Marketing Strategies," *Journal of Marketing*, vol. 21, no. 1, pp. 3–8, 1956. [Online]. Available: https://doi.org/10.2307/1247695

[5] ComplyRadar, "What happens when your AML transaction monitoring rules are static," 2020. [Online]. Available: https://complyradar.com/2020/03/10/what-happens-when-your-aml-transaction-monitoring-rules-are-static/

[6] S. Wang, L. Sun, and Y. Yu, "A dynamic customer segmentation approach by combining LRFMS and multivariate time series clustering," *Scientific Reports*, vol. 14, 07 2024. [Online]. Available: https://doi.org/10.1038/s41598-024-68621-2

[7] S. Mu and M. Punniyamoorthy, "Modified dynamic fuzzy c-means clustering algorithm – Application in dynamic customer segmentation," *Applied Intelligence*, vol. 50, 06 2020. [Online]. Available: https://doi.org/10.1007/s10489-019-01626-x

[8] M. Nordlinder, "Clustering of Financial Account Time Series Using Self Organizing Maps," Master's Thesis, KTH Royal Institute of Technology, Stockholm, Sweden, 2021.

[9] A. Eftekhari, L. Gaedke-Merzhäuser, D. Pasadakis, M. Bollhöfer, S. Scheidegger, and O. Schenk, "Algorithm 1042: Sparse Precision Matrix Estimation with SQUIC," *ACM Transactions on Mathematical Software*, vol. 50, no. 2, pp. 1 – 18, 2024. [Online]. Available: https://doi.org/10.1145/3650108

[10] D. Pasadakis, M. Bollhöfer, and O. Schenk, "Sparse Quadratic Approximation for Graph Learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 9, pp. 11 256–11 269, 2023. [Online]. Available: https://doi.org/10.1109/TPAMI.2023.3263969

[11] U. Luxburg, "A Tutorial on Spectral Clustering," *Statistics and Computing*, vol. 17, pp. 395–416, 01 2004. [Online]. Available: https://doi.org/10.1007/s11222-007-9033-z

[12] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008. [Online]. Available: https://doi.org/10.1088/1742-5468/2008/10/p10008

[13] V. A. Traag, L. Waltman, and N. J. van Eck, "From Louvain to Leiden: guaranteeing well-connected communities," *Scientific Reports*, vol. 9, no. 1, 2019. [Online]. Available: https://doi.org/10.1038/s41598-019-41695-z

[14] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, p. 226–231.

[15] A. Dempster, F. Petitjean, and G. Webb, "ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels," *Data Mining and Knowledge Discovery*, vol. 34, pp. 1454–1495, 2020. [Online]. Available: https://doi.org/10.1007/s10618-020-00701-z

[16] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, pp. 917, 963, 07 2019. [Online]. Available: https://doi.org/10.1007/s10618-019-00619-1

[17] T. Bester and B. Rosman, "Towards Financially Inclusive Credit Products Through Financial Time Series Clustering," 2024. [Online]. Available: https://arxiv.org/abs/2402.11066

[18] E. S. R.D. Wright, "How AML analytics can transform financial crime programs," 2022, last Accessed: 2025-09-07. [Online]. Available: https://www.crowe.com/insights/how-aml-analytics-can-transform-financial-crime-programs

[19] E. Dagum, "Time Series Modelling and Decomposition," *Statistica*, vol. 70, 05 2013. [Online]. Available: https://doi.org/10.6092/issn.1973-2201/3597

[20] R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning, "STL: A Seasonal-Trend Decomposition Procedure Based on Loess," *Journal of Official Statistics*, vol. 6, pp. 3–73, 1990.

[21] R. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 3rd ed. Australia: OTexts, 2021. [Online]. Available: https://otexts.com/fpp3/

[22] E. Dagum and S. Bianconcini, *Seasonal Adjustment Methods and Real Time Trend-Cycle Estimation*, ser. Statistics for Social and Behavioral Sciences. Springer International Publishing, 2016. [Online]. Available: https://doi.org/10.1007/978-3-319-31822-6

[23] S. Wierzchoń and M. A. Kłopotek, *Modern Algorithms of Cluster Analysis*, 1st ed., ser. Studies in Big Data. Springer Cham, 2018, vol. 34. [Online]. Available: https://doi.org/10.1007/978-3-319-69308-8

[24] C. Veenman, M. Reinders, and E. Backer, "A cellular coevolutionary algorithm for image segmentation," *IEEE Transactions on Image Processing*, vol. 12, pp. 304–316, 2003. [Online]. Available: https://doi.org/10.1109/TIP.2002.806256

[25] H. Frigui and R. Krishnapuram, "A robust competitive clustering algorithm with applications in computer vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 5, pp. 450–465, 1999. [Online]. Available: https://doi.org/10.1109/34.765656

[26] Y. Leung, J.-S. Zhang, and Z.-B. Xu, "Clustering by scale-space filtering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 12, pp. 1396–1410, 2000. [Online]. Available: https://doi.org/10.1109/34.895974

[27] M. Lechekhab, D. Pasadakis, and O. Schenk, "Multilevel Diffusion Based Spectral Graph Clustering," *28th Annual IEEE High Performance Extreme Computing Virtual Conference*, forthcoming. [Online]. Available: https://doi.org/10.1109/HPEC62836.2024.10938528

[28] R. Sanchez-Garcia, M. Fennelly, S. Norris, N. Wright, G. Niblo, J. Brodzki, and J. Bialek, "Hierarchical Spectral Clustering of Power Grids," *Power Systems, IEEE Transactions on*, vol. 29, pp. 2229–2237, 09 2014. [Online]. Available: https://doi.org/10.1109/TPWRS.2014.2306756

[29] A. A. Taha and A. Hanbury, "Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool," *BMC Medical Imaging*, vol. 15, p. 29, 2015. [Online]. Available: https://doi.org/10.1186/s12880-015-0068-x

[30] D. K. Singh and P. Choudhury, "Community detection in large-scale real-world networks," in *Principles of Big Graph: In-depth Insight*, ser. Advances in Computers. Elsevier, 2023, vol. 128, ch. 13, pp. 329–352. [Online]. Available: https://doi.org/10.1016/bs.adcom.2021.10.007

[31] X. Liu, H.-M. Cheng, and Z.-Y. Zhang, "Evaluation of Community Detection Methods," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 9, p. 1736–1746, 2020. [Online]. Available: https://doi.org/10.1109/TKDE.2019.2911943

[32] O. Kallenberg, *Foundations of Modern Probability*, 2nd ed., ser. Probability and Its Applications. New York, NY: Springer, 2002. [Online]. Available: https://doi.org/10.1007/978-1-4757-4015-8

[33] J. Friedman, T. Hastie, and R. Tibshirani, "Sparse inverse covariance estimation with the graphical lasso," *Biostatistics*, vol. 9, no. 3, pp. 432–441, 12 2007. [Online]. Available: https://doi.org/10.1093/biostatistics/kxm045

[34] C.-J. Hsieh, M. A. Sustik, I. S. Dhillon, and P. Ravikumar, "Sparse inverse covariance matrix estimation using quadratic approximation," in *Proceedings of the 25th International Conference on Neural Information Processing Systems*. Curran Associates Inc., 2011, p. 2330–2338. [Online]. Available: https://dl.acm.org/doi/10.5555/2986459.2986719

[35] C.-J. Hsieh, M. A. Sustik, I. S. Dhillon, P. Ravikumar, and R. A. Poldrack, "BIG & QUIC: sparse inverse covariance estimation for a million variables," in *Proceedings of the 27th International Conference on Neural Information Processing*

Systems - Volume 2. Curran Associates Inc., 2013, p. 3165–3173. [Online]. Available: https://dl.acm.org/doi/10.5555/2999792.2999965

[36] A. Eftekhari, L. Gaedke-Merzhäuser, D. Pasadakis, M. Bollhöfer, S. Scheidegger, and O. Schenk, "SQUIC user manual," 2023. [Online]. Available: https://www.gitlab.ci.inf.usi.ch/SQUIC/gitlab-profile/-/blob/main/

[37] M. Slawski and M. Hein, "Estimation of positive definite M-matrices and structure learning for attractive Gaussian Markov random fields," *Linear Algebra and its Applications*, vol. 473, pp. 145–179, 2015, special issue on Statistics. [Online]. Available: https://doi.org/10.1016/j.laa.2014.04.020

[38] E. A. Lopez-Rojas, A. Elmir, and S. Axelsson, "Paysim: A financial mobile money simulator for fraud detection," 2016.

[39] T. Suzumura and H. Kanezashi, "Anti-money laundering datasets: Inpluslab anti-money laundering datasets," 2021, last Accessed: 2025-04-12. [Online]. Available: http://github.com/IBM/AMLSim/

[40] M. Weber, J. Chen, T. Suzumura, A. Pareja, T. Ma, H. Kanezashi, T. Kaler, C. E. Leiserson, and T. B. Schardl, "Scalable Graph Learning for Anti-Money Laundering: A First Look," 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1812.00076

[41] V. Grimm, U. Berger, F. Bastiansen, S. Eliassen, V. Ginot, J. Giske, J. Goss-Custard, T. Grand, S. K. Heinz, G. Huse, A. Huth, J. U. Jepsen, C. Jørgensen, W. M. Mooij, B. Müller, G. Pe'er, C. Piou, S. F. Railsback, A. M. Robbins, M. M. Robbins, E. Rossmanith, N. Rüger, E. Strand, S. Souissi, R. A. Stillman, R. Vabø, U. Visser, and D. L. DeAngelis, "A standard protocol for describing individual-based and agent-based models," *Ecological Modelling*, vol. 198, no. 1, pp. 115–126, 2006. [Online]. Available: https://doi.org/10.1016/j.ecolmodel.2006.04.023

[42] S. Luke, C. Cioffi, L. Panait, K. Sullivan, and G. Balan, "MASON: A Multiagent Simulation Environment." *Simulation*, vol. 81, pp. 517–527, 2005. [Online]. Available: https://doi.org/10.1177/0037549705058073

[43] N. Rokotyan, O. Stukova, D. Kolmakova, and D. Ovsyannikov, "Cosmograph: Gpu-accelerated force graph layout and rendering," 2022. [Online]. Available: https://cosmograph.app/

[44] T. Manz, N. Abdennur, and N. Gehlenborg, "anywidget: reusable widgets for interactive analysis and visualization in computational notebooks," *Journal of Open Source Software*, vol. 9, no. 102, p. 6939, 2024, publisher: The Open Journal. [Online]. Available: https://doi.org/10.21105/joss.06939

[45] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967. [Online]. Available: https://doi.org/10.1109/TIT.1967.1053964