FEDERAL STATE AUTONOMOUS EDUCATIONAL INSITUTION FOR HIGHER PROFESSIONAL EDUCATION NATIONAL RESEARCH UNIVERSITY

«HIGHER SCHOOL OF ECONOMICS»

Faculty of Computer Science

Elfat Sabitov

Full Name

Топологический Метод Кластеризации Графов

Russian title

Topologically Inspired Graph Clustering

English title

Master of Science Dissertation

Field of study 01.04.02 «Applied Mathematics and Informatics»

Program: Math of Machine Learning

Student

Elfat Sabitov

Supervisor

Andrey Savchenko

Doctor of Engineering Sciences

Moscow, 2025

Topologically Inspired Graph Clustering

Elfat Sabitov

Submitted to the Skolkovo Institute of Science and Technology on June 11, 2025

ABSTRACT

Graph clustering is a common problem in diverse domains, such as social network analysis, computational biology, and logistics. Traditional methods often rely on geometric or statistical assumptions, while this thesis introduces a novel graph clustering algorithm inspired by topological data analysis (TDA), leveraging the concept of filtration to hierarchically explore graph connectivity and optimize clustering quality.

The proposed method constructs a Vietoris-Rips filtration to generate a sequence of simplicial complexes, then applies a modified clique percolation technique to detect clusters as dense regions of triangles (2-simplices). By analyzing the evolution of clusters across filtration thresholds, the algorithm directly optimizes clustering quality metric. This approach uniquely treats the filtration process as a search space for clustering optimization, a novel view on graph clustering.

The algorithm is validated on real-world benchmarks, including city road networks and a Vehicle Routing Problem (VRP) for taxi dispatch logistics. Experimental results demonstrate competitive performance with state-of-the-art algorithms (Louvain, Leiden) on synthetic and city graphs, while achieving superior cost reduction in VRP applications. Key innovations include the integration of filtration dynamics into modularity optimization and domain-specific validation on logistics challenges, highlighting its practical efficacy.

Keywords: graph clustering, Vietoris-Rips filtration, simplicial complex, vehicle routing problem

Research advisor:
Name: Ivan Oseledets
Degree, title: PhD, Professor

Co-advisor:
Name: Andrey Savchenko
Degree, title: PhD, Research scientist

Топологический Метод Кластеризации Графов

Эльфат Сабитов

Представлено в Сколковский институт науки и технологий
Июнь 11

Аннотация

Кластеризация графов является распространённой задачей в различных областях, таких как анализ социальных сетей, вычислительная биология и логистика. В отличие от традиционных методов, которые часто основываются на геометрических или статистических предположениях, данная работа представляет новый алгоритм кластеризации графов, основанный, который использует концепцию фильтрации для иерархического исследования связности графа и оптимизации качества кластеризации, позаимствованную из сферы топологического анализа данных (ТАД).

Предложенный метод строит фильтрацию Виеториса–Рипса для генерации последовательности симплициальных комплексов, после чего строит граф клик и ишет кластеры как области сильной связности, состоящие из треугольников (2-симплексов). Анализируя эволюцию кластеров при различных порогах фильтрации, алгоритм напрямую оптимизирует выбранную метрику качества кластеризации (модулярность). Данный подход уникальным образом рассматривает процесс фильтрации как пространство поиска для оптимизации кластеризации, что представляет собой новый взгляд на кластеризацию графов в целом.

Алгоритм проверен на реальных примерах, включая городские дорожные сети и задачу маршрутизации транспортных средств (VRP) для логистики распределения такси. Экспериментальные результаты демонстрируют конкурентоспособную производительность по сравнению с передовыми алгоритмами (Louvain, Leiden) на синтетических и городских графах, а также превосходное снижение затрат в приложениях VRP. Ключевые инновации включают интеграцию динамики фильтрации в оптимизацию модулярности и валидацию на задачах логистики, подчёркивая практическую эффективность метода.

Contents

Chapter 1
Introduction

Graph clustering is a fundamental problem in fields such as social network analysis, computational biology, and recommendation systems. For instance, music streaming platforms leverage clustering to group users with similar tastes, enabling personalized recommendations. Formally, the task involves partitioning a graph into subgraphs where vertices share similar properties. Existing approaches range from simple methods such as K-Medoids [14] to advanced modularity-based algorithms such as Louvain [1] and its successor Leiden [19]. In this work, clique percolation [13] serves as the foundation for the proposed novel clustering algorithm.

Topological Approach. Traditional clustering methods often rely on geometric or statistical assumptions (e.g., convexity, uniform density), which may not capture the inherent structure of complex graphs. Topological Data Analysis (TDA) addresses this limitation by computing topological invariants (e.g., connected components, loops) to reveal persistent patterns in data. For example, the ToMaTo clustering algorithm [2] identifies stable clusters using persistent homologies, while the MAPPER algorithm [17] constructs a simplicial complex from a user-defined filter function to highlight multiscale features.

Filtration clustering. Our approach diverges from these methods by focusing on the filtration process itself. Rather than computing persistence, we analyze the evolution of clusters across filtration timestamps to determine the optimal partitioning with respect to modularity 3.1. This strategy leverages the hierarchical information embedded in filtrations while avoiding the computational overhead of persistent homology.

Applications. Graph clustering has practical implications for NP-hard optimization problems, such as the Vehicle Routing Problem (VRP). VRP can be formulated as a Linear Programming (LP) problem (Eq. 5.3) and is widely used in logistics (e.g., cargo delivery, ride-sharing). Due to its computational complexity, clustering is employed to decompose large problems into smaller, tractable subproblems. However, the quality of the solution depends critically on the clustering algorithm. In this work, we evaluate our algorithm on real-world VRP dataset to validate its performance. This approach is not new; for example, [4] benchmarked clustering methods for VRP with time windows, demonstrating their impact on solution optimality.

Scientific Novelty. While TDA-based clustering is not new, our work introduces the following innovations:

- **Filtration as search space**: We propose the first method to directly optimize modularity over a filtration
- **VRP-specific validation**: We demonstrate superior performance on a real-world logistics benchmark.

Chapter 2
Author contribution

The author of this project researched the existing approaches for graph clustering, including those which use topological data analysis. After that, the algorithm was designed. The implementation includes:

- Novel graph clustering algorithm - Filtration Clustering.
- Experiments carried out on transport graphs of real cities and their visualization.
- Implementation of VRP benchmarking.

The results are stored in GitHub.

Chapter 3
Literature review

Graph clustering has been extensively studied across disciplines, from social network analysis to computational biology, driven by the need to uncover latent structures in complex systems. While traditional algorithms focus on geometric or statistical properties, recent advances leverage topological invariance to capture multiscale relationships. This section reviews classical and topology-inspired approaches, highlighting their strengths and limitations, ultimately positioning our work at the intersection of graph-native filtrations and modularity optimization.

KMedoids. One of the simplest approaches in graph clustering is, without doubt, **KMedoids** [14]. It's aim is to find k centroids inside graph structure. However, here lies it's main disadvantage - the need to explicitly define the number of clusters.

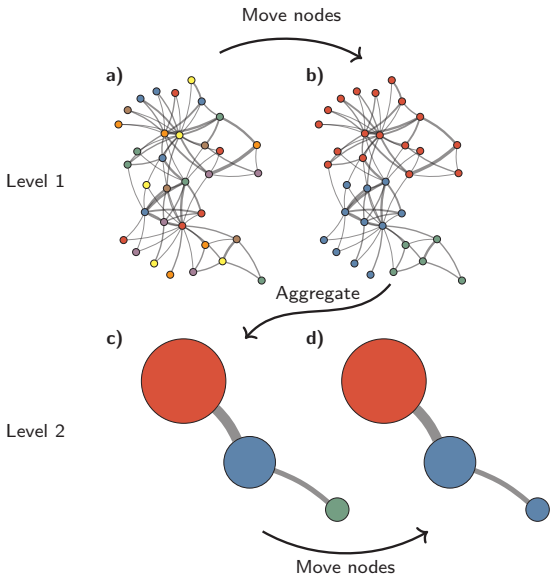


Figure 3.1: Louvain from [19].

Modularity based clustering. More advanced algorithms are built as optimization strategies with aim to optimize a common clustering metric - **Modularity** [3]:

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j) \quad (3.1)$$

Here A_{ij} - weight between nodes i and j; m - number of edges; k_i - degree of node i; $\delta(c_i, c_j)$ - the Kronecker delta function (1 if i and j belong to same cluster); γ - resolution parameter (higher gamma leads to larger number of smaller communities). This metric is widely used to measure clustering quality. It favors clusterings with strong connectivity inside clusters (a lot of edges between nodes of same clusters) and weak between them (few edges between nodes of different clusters). This metric is optimized in **Louvain** algorithm [1], which showed promising performance. It's core idea: build local communities and then aggregate clusters into new network (Fig. 3.1). At first, Louvain treats each node as an individual community, which is merged afterwards in such a way that modularity is maximized 3.1. However, this algorithm also possesses a critical drawback - sometimes it may produce disconnected communities.

To tackle the issues of Louvain, it's successor was made - **Leiden** algorithm [19]. In it's core it shares the same idea as the Louvain, however, it has theoretical guarantees that communities will be connected. In addition to that, authors added the opportunity to optimize another metric, called the **Constant Potts Model** (CPM) [18]:

$$H = -\sum_{i,j} \left(A_{ij} w_{ij} - \gamma \right) \delta(c_i, c_j) \quad (3.2)$$

Here analogously A_{ij} - weight between nodes i and j; $\delta(c_i, c_j)$ - the Kronecker delta function (1 if i and j belong to same cluster, 0 otherwise); γ - resolution parameter. Main difference of CPM is the fact that it better detects small communities. In this work, Leiden algorithm is considered as state of the art algorithm, universally used by researchers in various fields of graph analysis.

Topological approach A promising direction in data analysis explores the intrinsic topology of data through **Topological Data Analysis (TDA)**. Originating from computational topology, TDA aims to characterize data by its **topological invariants** (see Methodology 5 for details). Several attempts to apply TDA to graph clustering have already been made, with notable examples being **MAPPER** and **ToMaTo**.

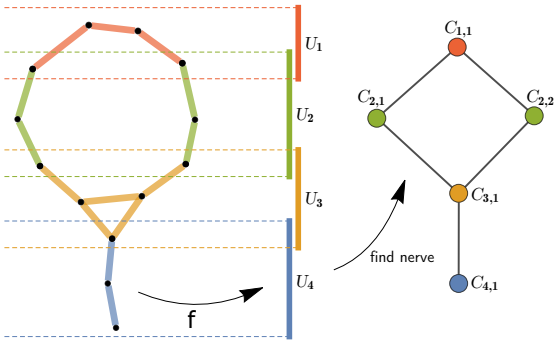


Figure 3.2: Mapper visualization from [10].

The **MAPPER** algorithm [17] employs a three-step pipeline: (1) project nodes to scalar values using a filter function (e.g., node degree, centrality), (2) cluster these values into overlapping intervals, and (3) construct a nerve (aggregated graph) from the clusters 3.2. While useful for exploratory analysis, MAPPER's performance is heavily dependent on the choice of filter function.

ToMATo [2] combines density estimation with Morse theory: nodes are assigned to scalar "heights" with estimators like Gaussian kernels or Distance-to-Measure (DTM). Starting at local maxima (density peaks), clusters are iteratively merged with neighboring lower-density regions until a single cluster remains. This process mirrors *Morse filtration*, where sublevel sets evolve as the density threshold decreases. For point clouds, ToMATo uses k-nearest neighbor or δ-Rips graphs (connecting nodes within radius δ). Like MAPPER, ToMATo's performance depends on the density estimator and scale parameter δ, requiring multi-scale evaluation for robustness.

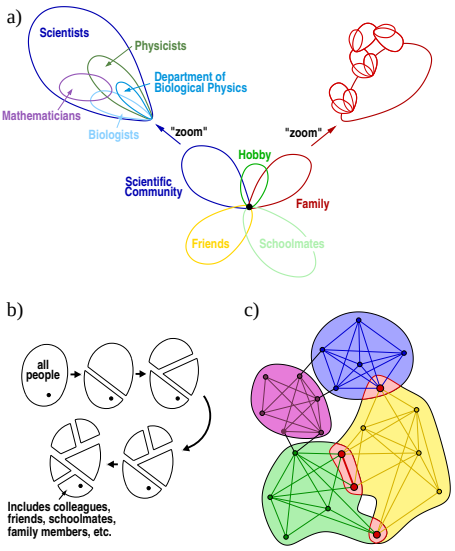


Figure 3.3: Visualization of clique percolation from [13].

Filtration Clustering Rather than relying on external filter functions, we propose a direct topological approach: we generate a sequence of δ-Rips graphs by thresholding edge weights and treat this filtration as a search space to identify the graph scale that optimizes clustering quality. This method eliminates filter function bias and leverages the inherent topology of edge weights for parameter-free clustering.

The clustering itself is performed with **Clique Percolation Method** [13].This method treats each k-clique as separate cluster and merges cliques if they are adjacent (Fig. 3.3). In our case, only 3-cliques are processed.

Chapter 4
Problem statement

Let $G(V, E, w)$ be an undirected weighted graph:

- $V = \{1, \ldots, n\}$: Set of vertices (nodes)
- $E \subseteq V \times V$: Set of edges
- $A \in \mathbb{R}^{n \times n}$: adjacency matrix where $A_{ij} = w_{ij}$ if $(i, j) \in E$, else 0

A *clustering* C partitions V into disjoint subsets $\{C_1, \ldots, C_k\}$.

Filtration Clustering. We generate a series of Vietoris-Rips graphs via a **filtration** 5.1:

$$\{G_\delta\}_{\delta \geq 0} = \{G(V, E_\delta, w_\delta) |\ w(e) \leq \delta \ \forall e \in E'\} \tag{4.1}$$

The task is to generate clusterings for each graph C_δ and choose the optimal one:

$$C^* = \arg\max_{C_\delta} \ \mathcal{Q}(C_\delta), \tag{4.2}$$

with $\mathcal{Q}$ as either Modularity 3.1 or Constant Potts Model (CPM) 3.2.

Chapter 5
Methodology

The core idea of our method relies on the concept of filtration, which is actively used in the field of Topological Data Analysis. In addition to that, we apply the clique percolation method [13] to the filtration. First, we define a simplicial complex:

Definition 5.1 ***Simplicial complex*** *on a finite vertex V is a collection $\mathcal{K} \subset 2^V$ satisfying the properties:*

1. *if $\sigma \subset \mathcal{K}$ and $\sigma' \subset \sigma$ then $\sigma' \subset \mathcal{K}$*
2. *$\emptyset \in \mathcal{K}$*

Elements $\sigma \subset \mathcal{K}$ are called simplices. A simplex $\sigma = \{v_1, ..., v_k\}$ is called a $(k-1)$-dimensional simplex.

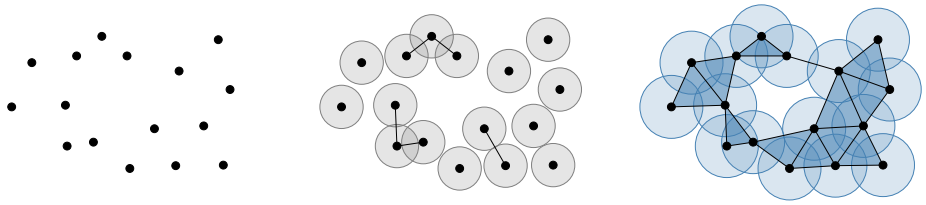
Examples:

- Vertices $\{v_i\}$ - 0-dim simplices
- Edges $\{v_i, v_j\}$ - 1-dim simplices
- Triangles $\{v_i, v_j, v_k\}$ - 2-dim simplices

Thus, simplicial complex is generalization of graph, which can be viewed as 1-skeleton - a simplicial complex, containing sumplices of dimension not greater than 1.

5.1 Vietoris-Rips Filtration

In the previous section, we have already mentioned Vietoris-Rips graph, which connects points if their distance is no more than δ. The Vietoris-Rips filtration constructs a generalization of this structure - multiscale simplicial complexes. Originally, these complexes are used for analysis of topological features of sampled space. The most common features are homologies $\mathcal{H}$, which in simple case represent connected components $\mathcal{H}_0$, loops $\mathcal{H}_1$ or cavities $\mathcal{H}_2$.5.1.



(a) ϵ_0 (b) ϵ_1 (c) ϵ_2

Figure 5.1: The Vietoris–Rips complex of a point cloud at different scales $\epsilon_0 < \epsilon_1 < \epsilon_2$. As the distance threshold ϵ increases, the connectivity changes and more edges emerge. Visualization from [9].

- **Vietoris-Rips Complex:** For a metric space (X, d) and the scale parameter $\varepsilon \geq 0$ the Vietoris-Rips Complex is the set of simplices σ:

$$VR_\varepsilon(X) = \{\sigma \subseteq X \mid \text{Diam}(\sigma) \leq 2\varepsilon\}$$

where $\text{Diam}(\sigma) = \max_{x,y \in \sigma} d(x, y)$. The simplices themselves are sets of

- **Filtration Structure:** A nested sequence of complexes:

$$VR_{\varepsilon_1} \hookrightarrow VR_{\varepsilon_2} \hookrightarrow \cdots \hookrightarrow VR_{\varepsilon_k} \quad \text{for } 0 \leq \varepsilon_1 < \varepsilon_2 < \cdots < \varepsilon_k$$

The main computational complexity of VR Complex is to find the exact moment, when high-dimensional simplices are born 5.2. Due to this issue, a **Čech complex** is usually computed:

$$VR_\varepsilon \subseteq \check{\text{C}}\text{ech}_{\varepsilon\sqrt{2}} \quad \text{and} \quad \check{\text{C}}\text{ech}_\varepsilon \subseteq VR_\varepsilon$$

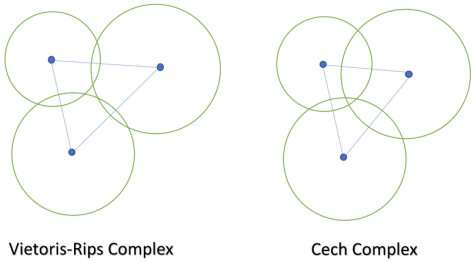


Figure 5.2: The difference between Vietoris-Rips Complex and Čech complex [6]

5.2 Filtration clustering

Both Leiden [19] and Louvain [1] treat clusterization as an optimization problem: find clusters $C_1 \cup ... \cup C_n = V$ such that they maximize some quality metric $F(G(V, E), \{C_i\})$ - either Modularity 3.1 or CPM 3.2. We propose treating Filtration structure as search space in this optimization problem.

Clique Percolation Method. First of all, we need to specify how the clusterization of Vietoris-Rips Complexes will be performed. Our main goal is to detect high-density regions, which form the core of clusters. To do that, we decided to treat 2-simplexes (2-cliques or triangles) in each generated simplicial complex VR_ε as clusters, regions of high density. Triangles with adjacent edges are treated as one cluster, points that do not belong to any triangle are assigned to the closest cluster. The described approach is the modified special case of Clique Percolation Method [13]. This method finds all k-cliques in the graph, treating each of them as a cluster, and merges them if they are adjacent by $(k-1)$-clique (boundary):

1. Get adjacency of k-cliques
2. Build graph of cliques, based on that adjacency
3. Find all connected components - these are the clusters, built on top of merged cliques.

Filtration clique graph. In our algorithm 1 we start with the original graph G. First, in this graph, we find all triangles $\{t_i\}_{i=1}^k$ and their birth times $\{d_i = \max_{e_i \in \partial t_i}(e_i)\}_{i=1}^k$ - largest weight on the boundary of triangle (largest edge). On top of it, we build the graph of triangles $G_t(V_t, E_t)$, where the nodes are triangles $V_t = \{t_i\}$ and the edges $E_t = \{(t_i, t_j) : |t_i \cap t_j| = 2\}$ exist if triangles share an edge. Now we find connected components of G_t, save them and move to the main part begins:

- We iterate over triangles, starting with those that have the largest birth times.
- Remove triangle from G_t - change it's topology.
- Find all connected components and save them.

We get a series of saved connected components, which are actually clusterings of the original graph. In case if $V \neq C_1 \cup ... \cup C_n$, we just assign unlabeled nodes to the closest clusters. Among these clusterings $\{C_i\}_\varepsilon$ the optimal one is chosen with respect to quality metric $F(G(V, E), \{C_i\})$.

Algorithm 1 Filtration clustering

Require: Graph $G(V, E)$, quality functional F
Ensure: $V = C_1 \cup ... \cup C_n$
  $\{(t_i, d_i)\}_{i=1}^k \leftarrow find\_all\_triangles(G)$
  $G_t(V_t = \{t_i\}, E_t) \leftarrow triangle\_adjacency(\{(t_i, d_i)\})$
  $best\_score \leftarrow -\inf$
  $best\_clustering \leftarrow 0$
  for $i = 0$ **to** k **do**
    if $i > 0$ **then**
      $G_t \leftarrow remove\_node(G_t, t_i)$
    end if
    $\{C_j\}_i^{base} \leftarrow connected\_components(G_t)$
    $\{C_j\}_i \leftarrow assign\_closest\_cluster(G, \{C_j\}_i^{base})$
    $clustering_i \leftarrow \{C_j\}_i$
    $score \leftarrow F(G(V, E), clustering_i)$
    if $score > best\_score$ **then**
      $best\_score \leftarrow score$
      $best\_clustering \leftarrow clustering_i$
    end if
  end for
  return $best\_clustering$

Parallel algorithm. The final version of our algorithm manages to utilize several processes for faster computation. First, we divide list of triangles $\{t_i\}$ into n parts, where n - number of available processes. Then we recursively prepare a set of graphs: $G_t^{q+1} = subgraph(G_t^q)$ - simply remove nodes from previous levels. Thus, we divide our filtration space into almost equal chunks, which we explore in parallel. Then we simply take the best result that our processes have found.

Border perturbations. After getting the final result of our algorithm, we may also slightly improve clustering by trying to assign nodes on the borders of clusters, to neighbouring clusters. This heuristic gives a slight improvement in modularity and CPM.

5.3 Vehicle Routing Problem

For benchmarking, the algorithm was applied to the Vehicle Routing Problem, based on real data. This is a Linear Programming problem with multiple formulation approaches.

Core Formulation (Arc-Based)

- **Variables:**
  - $x_{ij}^k \in \{0, 1\}$: Vehicle k travels from node i to node j.
  - $y_{ik} \in \{0, 1\}$: Customer i is served by vehicle k.
- **Objective:**

$$\text{Minimize} \quad \sum_{k=1}^{m} \sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} x_{ij}^k$$

Key Constraints

1. **Customer service:**

$$\sum_{k=1}^{m} y_{ik} = 1, \quad \forall i \in \{1, \ldots, n\}$$

2. **Depot operations:**

$$\sum_{j=1}^{n} x_{0j}^k = 1, \quad \sum_{i=1}^{n} x_{i0}^k = 1, \quad \forall k \in \{1, \ldots, m\}$$

3. **Flow conservation:**

$$\sum_{i=0}^{n} x_{ij}^k = y_{jk}, \quad \sum_{j=0}^{n} x_{ij}^k = y_{ik}, \quad \forall j \in \{1, \ldots, n\}, \forall k \in \{1, \ldots, m\}$$

4. **Capacity limits:**

$$\sum_{i=1}^{n} q_i y_{ik} \leq Q, \quad \forall k \in \{1, \ldots, m\}$$

Alternative Formulations

1. Step-Based Formulation

Instead of arc variables x_{ij}^k, use sequential routing variables:

- $x_{si}^k \in \{0, 1\}$: Vehicle k visits customer i at step s.
- Define a maximum route length S_{max}.

Advantages:

- Explicit route sequencing.
- Easier integration of temporal constraints.

New constraints:

1. **Step continuity:**

$$\sum_{s=1}^{S_{\max}-1} x_{si}^k \geq \sum_{s=2}^{S_{\max}} x_{si}^k, \quad \forall i \in \{1,\ldots,n\}, \forall k \in \{1,\ldots,m\}$$

2. **Single position per customer:**

$$\sum_{s=1}^{S_{\max}} \sum_{k=1}^{m} x_{si}^k = 1, \quad \forall i \in \{1,\ldots,n\}$$

2. Miller-Tucker-Zemlin (MTZ) Subtour Elimination

Previous formulation may generate subtours - cycles, which do not start at depot. To eliminate them more efficiently, continuous variables are introduced $u_i \geq 0$ representing the position of node i in the route:

$$u_i - u_j + n x_{ij}^k \leq n-1, \quad \forall i,j \in \{1,\ldots,n\}, \forall k \in \{1,\ldots,m\}$$

with bounds

$$1 \leq u_i \leq n, \quad \forall i \in \{1,\ldots,n\}$$

Benefits:

- Polynomial number of constraints.
- More computationally tractable than combinatorial subtour elimination.

Additional Considerations

Symmetry Breaking For homogeneous fleets, to reduce symmetric solutions:

$$\sum_{i=1}^{n} x_{0i}^k \geq \sum_{i=1}^{n} x_{0i}^{k+1}, \quad \forall k \in \{1,\ldots,m-1\}$$

Route Duration Limits If vehicles have a maximum operating time $T_{\max}$:

$$\sum_{i=0}^{n} \sum_{j=0}^{n} t_{ij} x_{ij}^k \leq T_{\max}, \quad \forall k \in \{1,\ldots,m\}$$

where t_{ij} is the travel time from node i to node j.

Model Selection Guidance

- **Arc-based formulation**: Best suited for exact solutions on small to medium-sized instances.
- **Step-based formulation**: Preferred for heuristic or metaheuristic approaches, especially when route sequencing is critical.
- **MTZ formulation**: Recommended for problems requiring efficient subtour elimination with manageable model size.

Chapter 6
Numerical experiments

All experiments have been conducted in Python. The packages used:

- To handle graph structures and operations on them (including Louvain algorithm), networkx framework [5].
- The majority of arithmetic operations were covered by numpy, sklearn and built-in python libraries
- Visualizations done with matplotlib.
- Parallel computations performed with joblib [7].
- Leiden implementation in C++ with leidenalg.
- KMedoids implementation in Rust [16].
- City graphs provided by [12].
- Vehicle Routing Problem solving done with Google OR-Tools and CP-SAT solver [15].

6.1 Graphs with ground truth

Our tests start with graphs with ground-truth communities. The experiments were carried out on a simple Zachary Karate Club [20] and a more complex network of email communications from Stanford Large Network Dataset Collection [8]. We compared how well do Louvain, Leiden and Filtration Clustering algorithms correlate with ground truth communities using two score functions:

- **Adjusted Mutual Information (AMI)**:

$$\text{AMI}(U, V) = \frac{\text{MI}(U, V) - E[\text{MI}(U, V)]}{\text{avg}(H(U), H(V)) - E[\text{MI}(U, V)]},$$

  where U - Ground truth clustering; V - Predicted clustering; $\text{MI}(U, V)$ - Mutual Information between U and V; $E[\text{MI}(U, V)]$- Expected MI under random chance; $H(U), H(V)$: Entropies of U and V; avg - Arithmetic, geometric, min, or max average (default: arithmetic)

- **Adjusted Rand Score (ARI)**:

$$\text{ARI} = \frac{\text{RI} - E[\text{RI}]}{\max(\text{RI}) - E[\text{RI}]},$$

  where RI - Rand Index, which counts agreement in pairwise sample assignments; $E[\text{RI}]$ - Expected RI under random chance.

Basically, AMI shows how much knowing the value of one variable reduces the uncertainty about the other, and ARI shows how much pairs coincide in two clusterings.

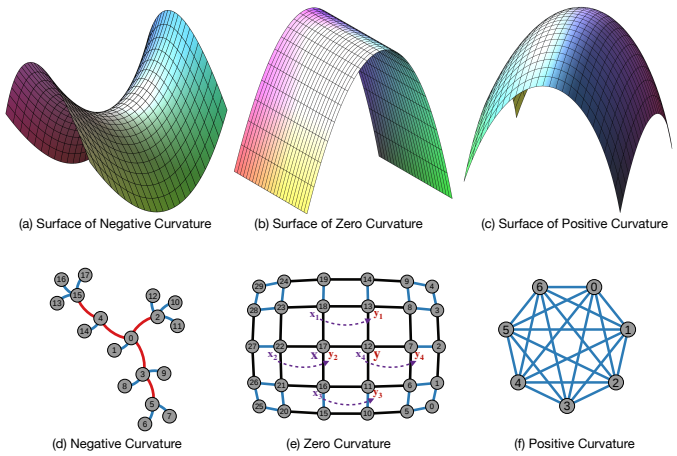


Figure 6.1: Visualization of graph curvatures from [11].

Curvature. It is common for some networks to be unweighted graphs. That was the case for network of email communications. However, our filtration process relied on weighted edges. To address this issue, we computed the graph curvature for each edge, a discrete analogue of Ricci Curvature [11]. Basically, the curvature is positive in regions of high density, zero in grid-like structures and negative in tree-like graphs 6.1.

Results. Measurements showed that algorithms have similar performance and relatively moderate level of correlation with ground truth labels. Perfect scores in this case are impossible due to potential noise in the labels themselves.

Table 6.1: Leiden, Louvain and Filtration Clustering on email netwrokx dataset.

Metric	Filtration	Leiden	Louvain
AMI	0.56454	0.56902	0.56665
ARI	0.33782	0.34366	0.35809
Modularity	0.31704	0.41745	0.41112

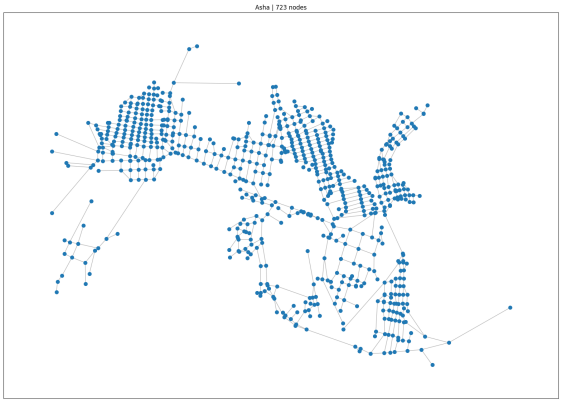
6.2 City Graphs

Before moving to VRP, we tested the performance of our algorithms on city graphs: Asha, Karaganda and Ekaterinburg 6.2. These graphs represent city roads, split into intervals - edges. Each edge has weight. representing distance in kilometers. Vertices are just points, where edges are glued (e.g. crossroads), each node has coordinates: latitude and longitude. Due to the grid-like nature of transport graphs, the amount of 2-cliques is almost zero. To address this issue, we processed original city graph and generated the graph of k-nearest neighbours with respect to the shortest path distances 6.2. As it can be seen, such approach already cuts graph into separate connected components, the choice of k mainly influences the connectivity of remote nodes.

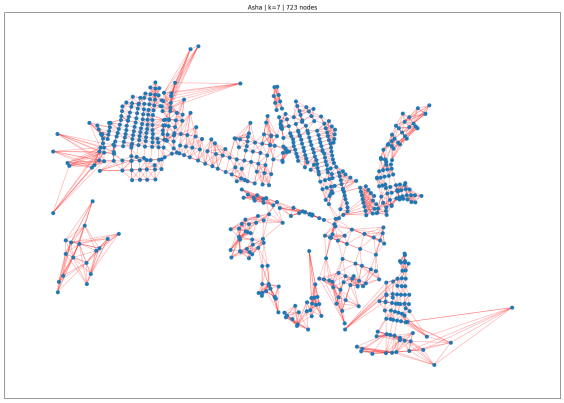
The baseline methods we chose here were Louvain and Leiden algorithms. Performance was measured in modularity 6.2. Our algorithm performed filtration and chose the optimal clustering from 100 thresholds and showed reasonable clusterings.

6.3 Vehicle Routing Problems Benchmark

To test the efficacy of the algorithm, we applied it to a real problem - taxi dispatch. Basically it is Vehicle Routing Problem with modification: we are given a set of requests $\{r_i\}$, where each request $r_i = (t_i, s_i, d_i)$ is represented by the request timestamp t_i, the pickup point s_i and the drop-off point d_i. The goal is to shrink the size of the set of $\{r_i\}$ by optimally merging requests with each other 6.5. For example, make people drive together if they share the starting point or pick up a person on

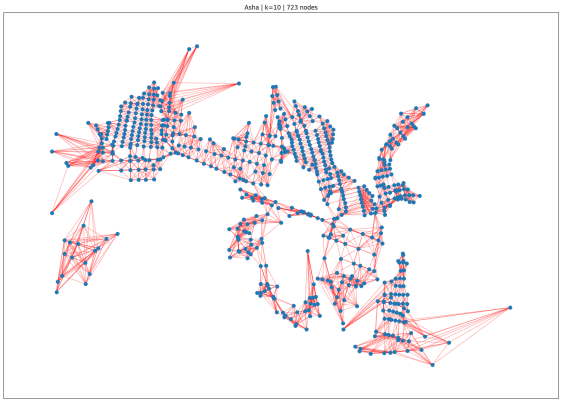


(a) Asha city

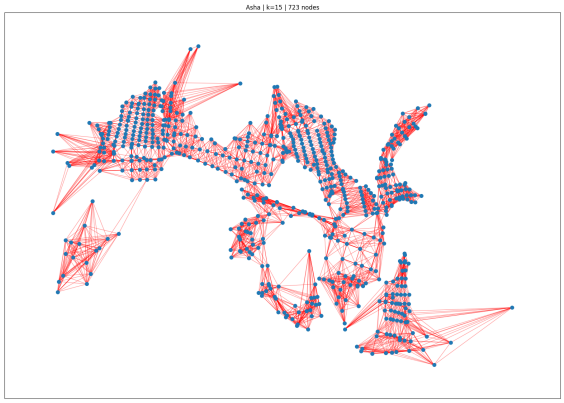


(b) Asha knn graph $k = 7$

Figure 6.2: Asha city (Chelyabinsk region) and it's knn graph for $k = 7$. knn graph contains separate connected components.



(a) Asha knn graph $k = 10$



(b) Asha knn graph $k = 15$

Figure 6.3: In case of $k = 10$ and $k = 15$ the connectivity evidently got far stronger.

Table 6.2: Clustering performance of Leiden, Louvain and Filtration clustering algorithm on Asha city and it's knn graph measured in modularity

City	Louvain	Leiden	Filtration
Asha	0.92057	0.92252	-
Asha $k = 3$	0.93838	0.93993	0.88109
Asha $k = 5$	0.91860	0.91879	0.87332
Asha $k = 7$	0.90426	0.90672	0.82745
Asha $k = 10$	0.88051	0.88541	0.76219
Asha $k = 15$	0.85497	0.85759	0.67763

the way. To do this optimally, we also fix the time window H within which the merge is considered feasible:

- **Input**: N requests $r_i = (t_i, s_i, d_i)$ where:
  - t_i: Request timestamp
  - s_i: Pickup location (lat/long)
  - d_i: Drop-off location

Table 6.3: Clustering performance of same algorithms on larger city graphs: Karaganda (Krg) with 3345 nodes and Ekaterinburg (Ekb) with 5608 nodes.

	Krg	Krg $k=3$	Krg $k=5$	Ekb	Ekb $k=3$	Ekb $k=5$
Louvain	0.96540	0.98248	0.97083	0.97114	0.98739	0.97610
Leiden	0.96612	0.98247	0.97118	0.97145	0.98757	0.97639
Filtration	-	0.91306	0.92137	-	0.92911	0.89945

- **Constraints**:
  - Temporal: Merge window H
  - Spatial: Network distance through OpenStreetMap

The problem in general can be solved with the help of Google OR-Tools. This library provides solvers for Linear Programming Problems and is very popular among the community of engineers designing custom logistic solutions. The key issue here is problem's complexity: for large VRPs more or less optimal solution may be found after a long period of time. To optimize the search, graph clustering can be applied.

First, requests are converted into a graph of deliveries $G(V, E, w)$, where the absence of an edge indicates that two requests cannot be merged due to the long time interval between creation:

- **Nodes**: Taxi requests $V = \{r_i\}$
- **Edges**: $E = \{(r_i, r_j) : |t_i - t_j| \leq H\}$
- **Weights**: $w_{ij} = d_{\text{network}}(d_i, d_j)$

This graph is clustered into a series of subgraphs, representing requests feasible for merge, which are passed to OR-tools solver. Thus, instead of solving a large optimization problem, we solve a set of smaller subproblems.

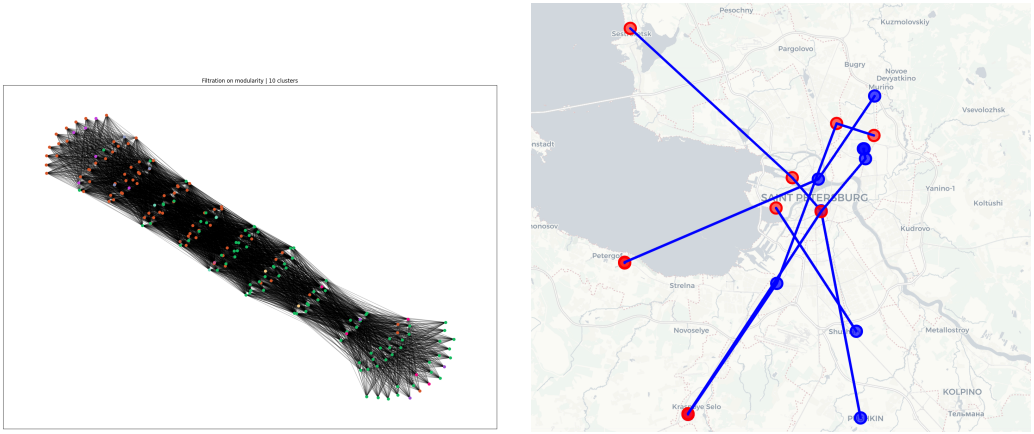
Synthetic data

First we tested our algorithm on synthetic data and created a pipeline for its generation:

1. Pick a city graph from OpenStreetMap.
2. Sample nodes from this graph as addresses.
3. Sample start and end points from addresses.
4. Randomly assign creation time within 24 hours.

We picked a graph of Saint-Petersburg and generated 200 requests, which were converted to a single graph of deliveries. The graph itself had one connected component, which we clustered using KMedoids, Louvain, Leiden and Filtration Clustering 6.4.

The key metric to be optimized in this problem is the total cost of all taxi requests combined, which is also linearly dependent on the route's distance. We tested our filtration algorithm against KMedoids, Louvain and Leiden algorithms 6.4. In this case, Leiden algorithm showed best performance, our algorithm was on the same level as K-medoids. The key problem with syntetic data - it is far from reality. Real delivery requests tend to cluster around specific points: either offices or crowded places.



(a) Filtration Clustering on deliveries graph

(b) Subset of solution routes

Figure 6.4: Clustering result of Filtration Clustering algorithm and subset of it's solution - sample of merged routes.

Table 6.4: Clustering performance of Leiden, Louvain and Filtration clustering algorithm measured in modularity

	Baseline	K-medoids	Louvain	Leiden	Filtration
N routes	200	179	170	**163**	180
Distance (km)	5040	4711	4627	**4144**	4784
Cost (rub)	429859	401328	395858	**356671**	408468

Real data

As for the real data, the visualizations 6.6 clearly show that there are two connected components in the delivery graph. This means that there are two disjoint sets of requests (e.g. morning and after-work hours), which are too far from each other in terms of the chosen time window $H = 2$ hours.

Once again, the key optimization metric is the total cost of all taxi requests combined. Key takeaways of the test 6.5:

- Clustering reduces route count by 31% vs baseline
- Filtration clustering performed comparably to modularity methods and achieved minimal total cost
- K-Medoids shows poorest spatial coherence

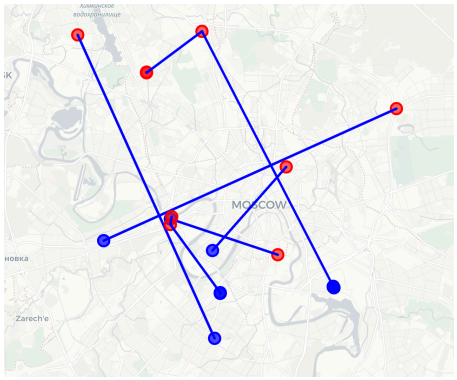
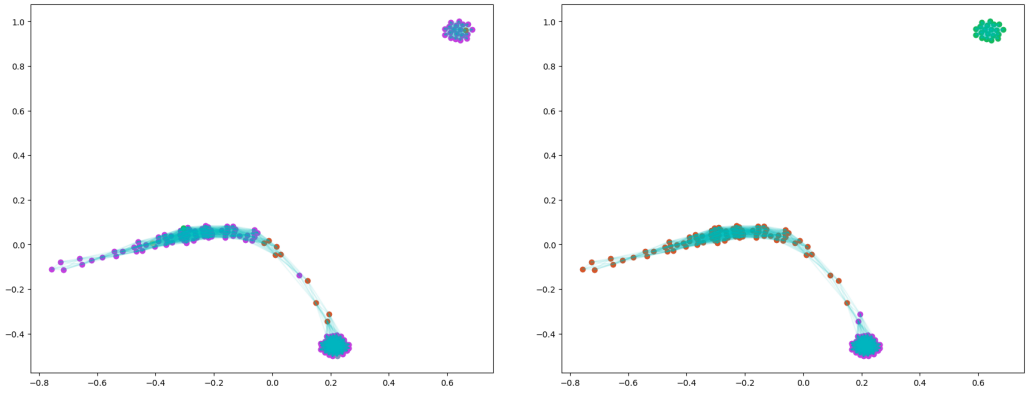


Figure 6.5: Subset of solution routes in Moscow: blue - start, red - finish.



(a) K-medoids (b) Filtration clustering

Figure 6.6: K-medoids with parameter $k = 5$ found 4 clusters and gave the worst performance, Filtration clustering found 3 clusters.

Table 6.5: Clustering performance of Leiden, Louvain and Filtration clustering algorithm measured in modularity

	Baseline	K-medoids	Louvain	Leiden	Filtration
N routes	146	102	98	99	**98**
Distance (km)	1636	1301	1287	1290	**1283**
Cost (rub)	153724	121799	120079	120321	**119823**

Chapter 7
Discussion and conclusion

This thesis advances the field of graph clustering by introducing a topologically inspired algorithm that leverages Vietoris–Rips filtrations and clique percolation to optimize clustering quality. The method offers a novel perspective by treating the graph filtration as a search space, enabling parameter-free hierarchical clustering that is both theoretically grounded and practically effective. Experimental validation on synthetic and real-world datasets, particularly in the context of vehicle routing, demonstrates the method's competitive performance and practical relevance. While challenges remain in terms of scalability and computational efficiency, the proposed approach lays the groundwork for future research at the intersection of topological data analysis and graph optimization.

Acknowledgements

Author thanks Dmitry Fedorov and Nickita Zakharenko for meaningful discussion of research ideas and help with solving Vehicle Routing Problem.

7.1 Usage of Generative AI

During the writing process author used Perplexity and DeepSeek to adjust the text style, correct grammar and improve readability. The final version of the thesis was carefully reviewed and edited.

Bibliography

[1] Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment 2008*, 10 (oct 2008), P10008.

[2] Chazal, F., Guibas, L., Oudot, S., and Skraba, P. Persistence-based clustering in riemannian manifolds. *Journal of the ACM 60* (06 2011).

[3] Clauset, A., Newman, M. E. J., and Moore, C. Finding community structure in very large networks. *Physical Review E 70*, 6 (Dec. 2004).

[4] göçken, T., and Yaktubay, M. Comparison of different clustering algorithms via genetic algorithm for vrptw. *International Journal of Simulation Modelling 18* (12 2019), 574–585.

[5] Hagberg, A., Swart, P., and S Chult, D. Exploring network structure, dynamics, and function using networkx. Tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[6] Jiang, B., Huang, Y., Panahi, A., Yu, Y., Krim, H., and Smith, S. L. Dynamic graph learning: A structure-driven approach. *Mathematics 9*, 2 (2021).

[7] Joblib Development Team. Joblib: running python functions as pipeline jobs, 2020.

[8] Leskovec, J., and Krevl, A. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[9] Moor, M., Horn, M., Rieck, B., and Borgwardt, K. Topological autoencoders, 2021.

[10] Murugan, J., and Robertson, D. An introduction to topological data analysis for physicists: From lgm to frbs, 2019.

[11] Ni, C.-C., Lin, Y.-Y., Luo, F., and Gao, J. Community detection on networks with ricci flow. *Scientific reports 9*, 1 (2019), 1–12.

[12] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2017.

[13] Palla G, Derényi I, F. I., and T, V. Uncovering the overlapping community structure of complex networks in nature and society. *Nature* (2005).

[14] Park, H.-S., and Jun, C.-H. A simple and fast algorithm for k-medoids clustering. *Expert Systems with Applications*.

[15] Perron, L., and Didier, F. Cp-sat.

[16] Schubert, E., and Lenssen, L. Fast k-medoids clustering in rust and python. *Journal of Open Source Software* (2022).

[17] Singh, G., Mémoli, F., and Carlsson, G. Topological methods for the analysis of high dimensional data sets and 3d object recognition. pp. 91–100.

[18] Traag, V. A., Van Dooren, P., and Nesterov, Y. Narrow scope for resolution-limit-free community detection. *Phys. Rev. E 84* (Jul 2011), 016114.

[19] Traag V.A., Waltman L., v. E. N. From louvain to leiden: guaranteeing well-connected communities. *Nature* (2019).

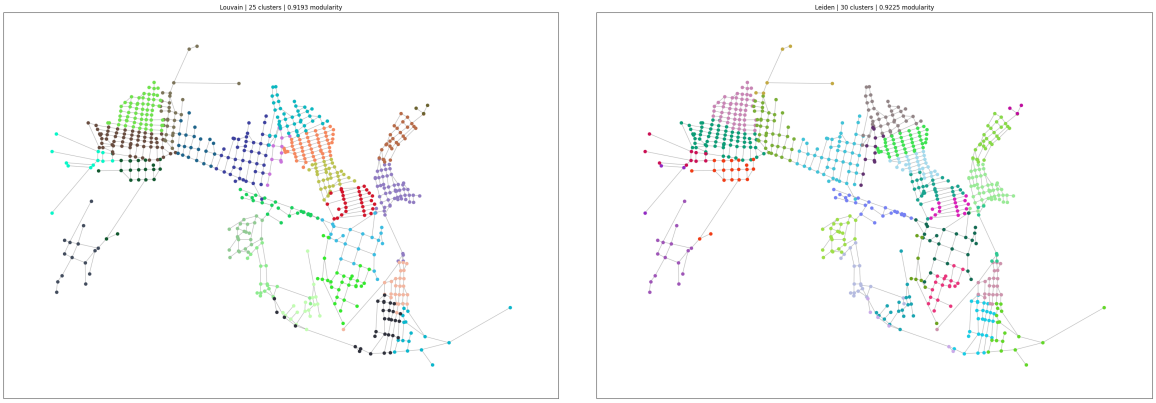
[20] Zachary, W. W. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research 33*, 4 (Dec. 1977), 452–473.

Appendix

7.2 City Graphs

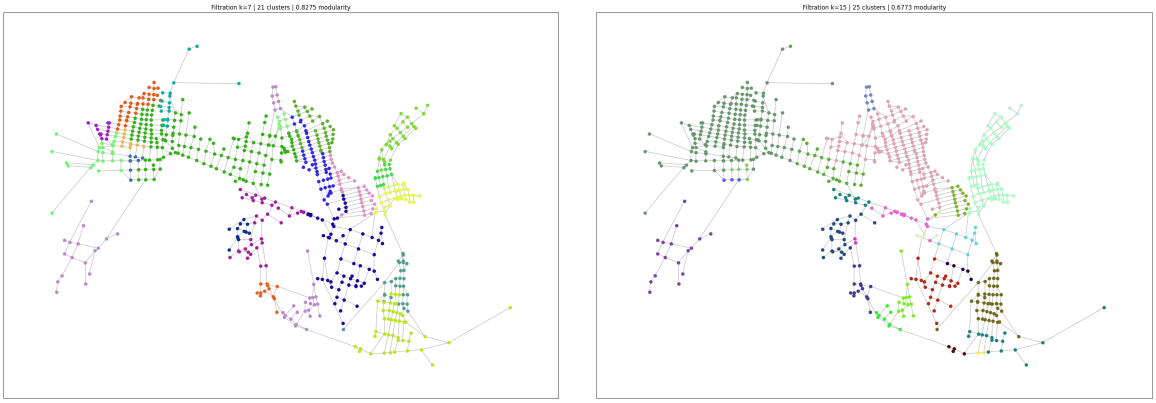
Here we present the visualizations of city clustering we performed with Louvain, Leiden and Filtration Clustering algorithms on Asha, Karaganda and Ekaterinburg.

Asha



(a) Louvain (b) Leiden

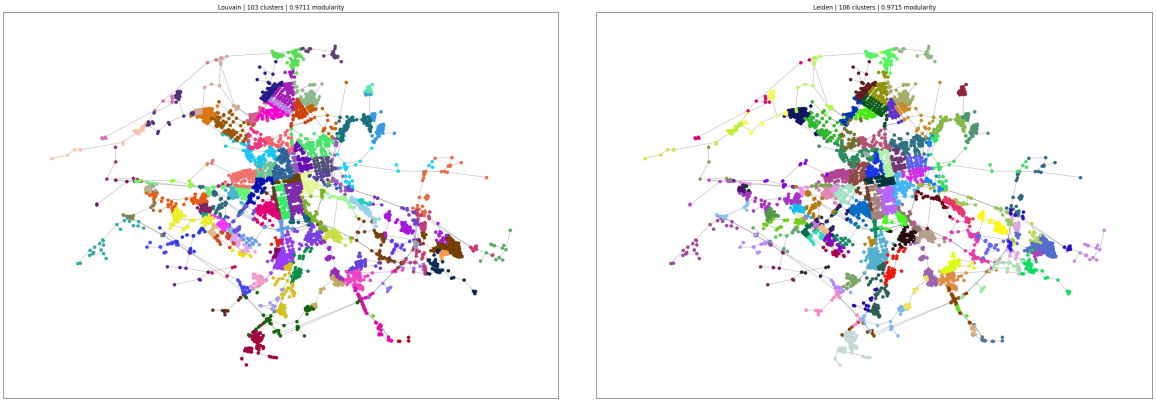
Figure 7.1: Result of Louvain and Leiden algorithm applied to Asha city



(a) Filtration $k = 7$ (b) Filtration $k = 15$

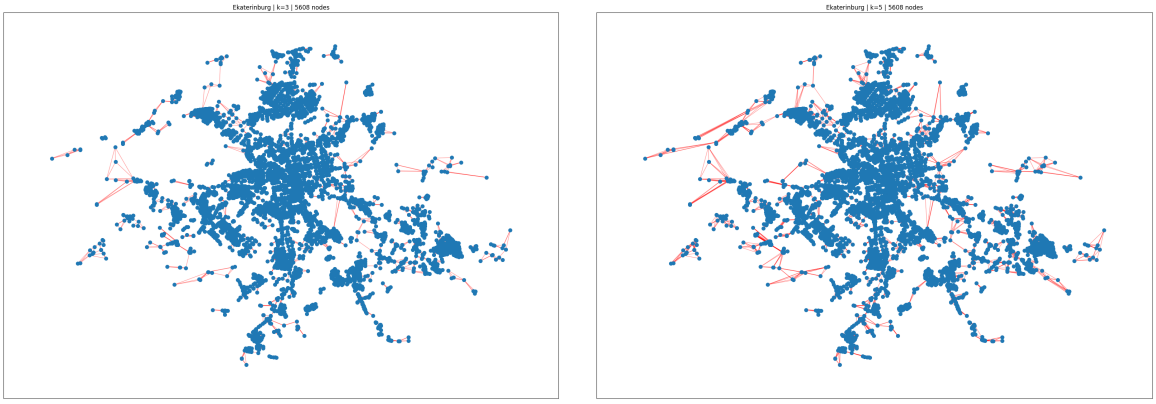
Figure 7.2: Result of Filtration Clustering on knn graph with $k = 7$ and $k = 15$

Ekaterinburg



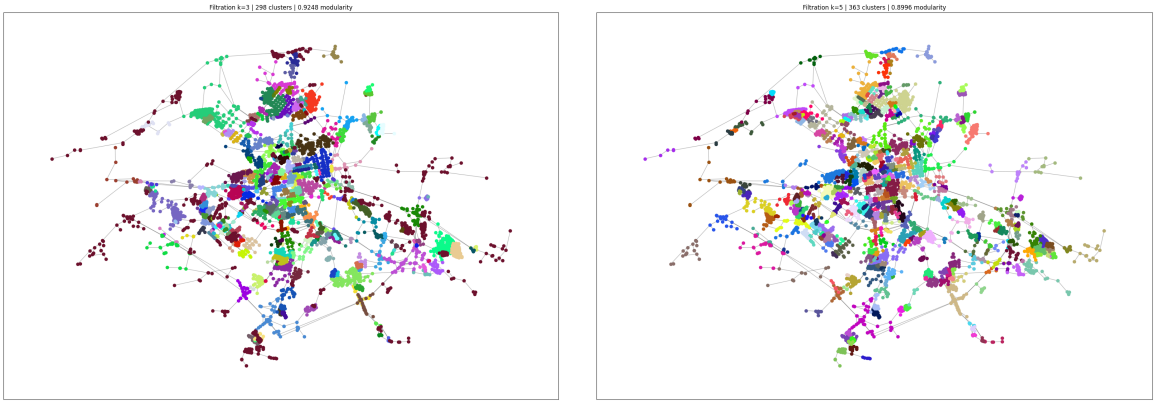
(a) Louvain (b) Leiden

Figure 7.3: Result of Louvain and Leiden algorithm applied to Ekaterinburg city



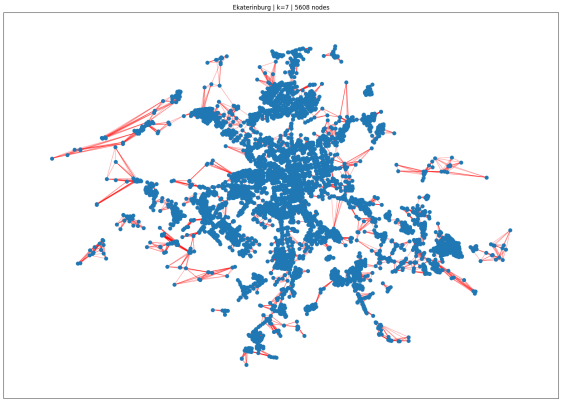
(a) knn graph $k = 3$ (b) knn graph $k = 5$

Figure 7.4: Knn graphs of Ekaterinburg, $k = 3$ and $k = 5$.

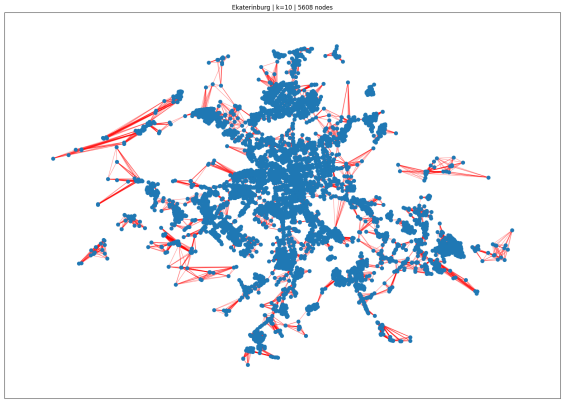


(a) Filtration on knn graph $k = 3$ (b) Filtration on knn graph $k = 5$

Figure 7.5: Filtration clustering on knn graphs of Ekaterinburg, $k = 3$ and $k = 5$.

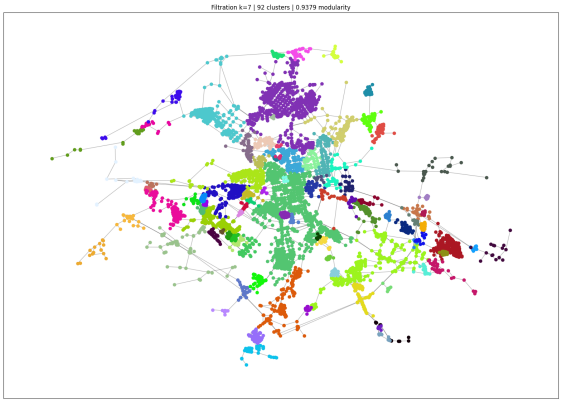


(a) knn graph $k = 7$

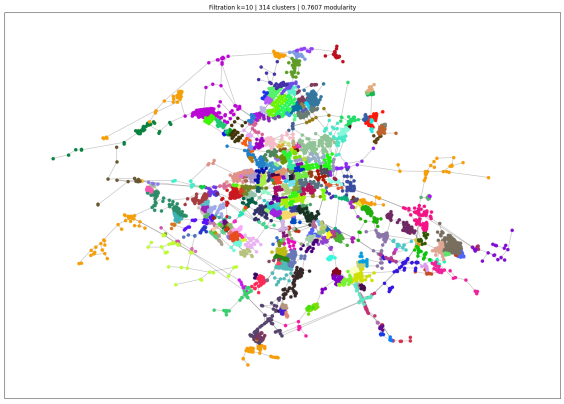


(b) knn graph $k = 10$

Figure 7.6: Knn graphs of Ekaterinburg, $k = 7$ and $k = 10$.

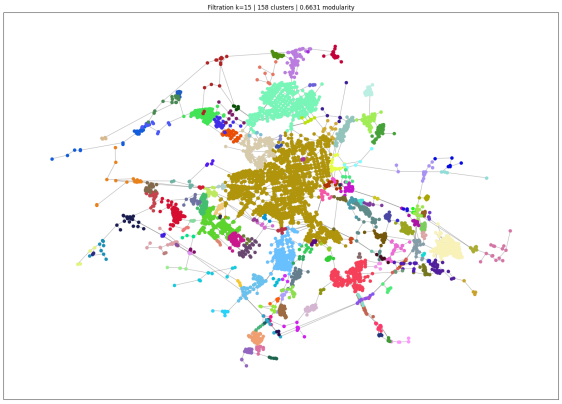


(a) Filtration on knn graph $k = 7$

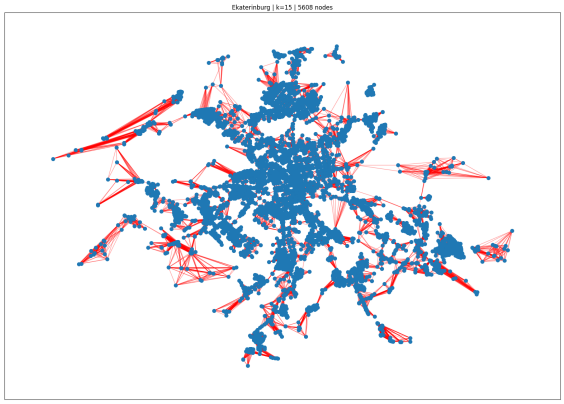


(b) Filtration on knn graph $k = 10$

Figure 7.7: Filtration clustering on knn graphs of Ekaterinburg, $k = 7$ and $k = 10$.



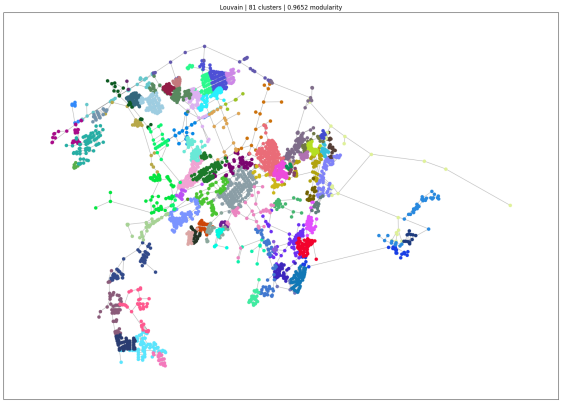
(a) Filtration on knn graph $k = 15$



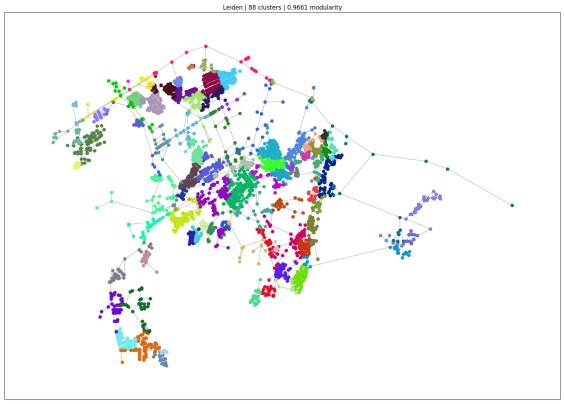
(b) knn graph $k = 15$

Figure 7.8: Knn graph of Ekaterinburg and Filtration clustering on it, $k = 15$.

Karaganda

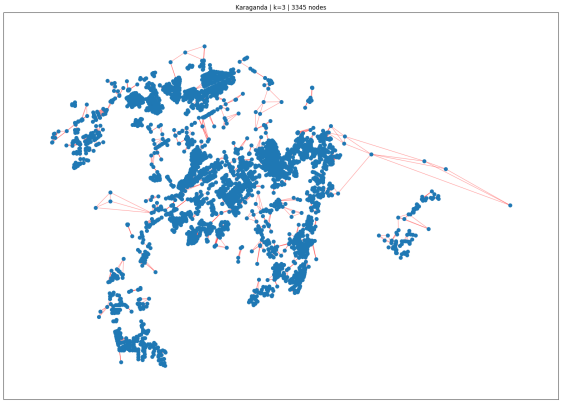


(a) Louvain

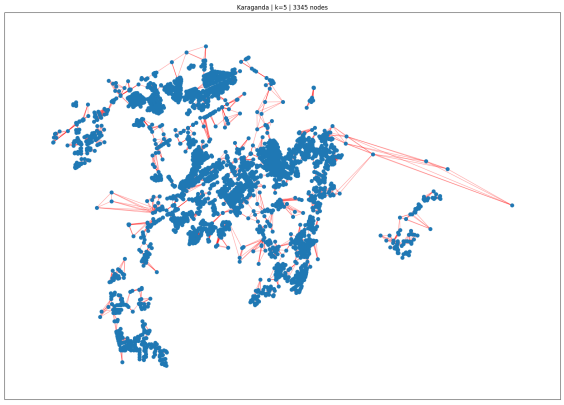


(b) Leiden

Figure 7.9: Result of Louvain and Leiden algorithm applied to Karaganda city

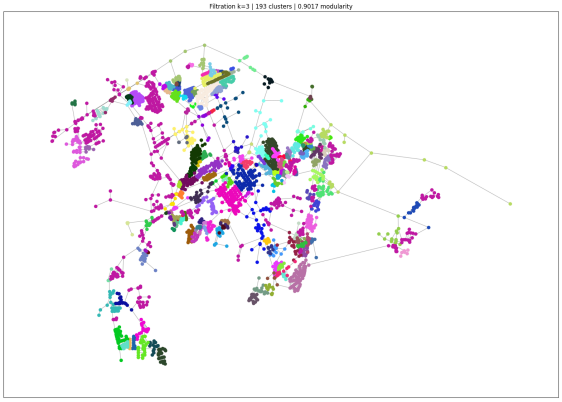


(a) knn graph $k = 3$

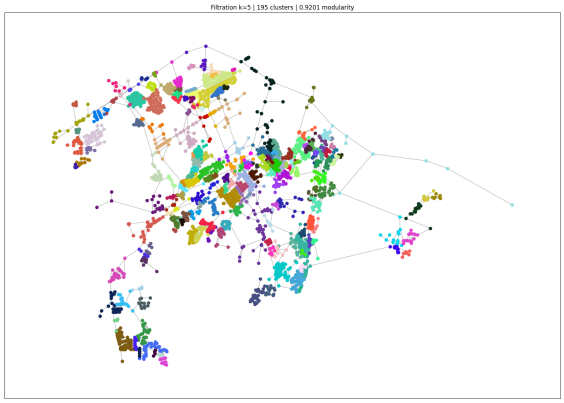


(b) knn graph $k = 5$

Figure 7.10: Knn graphs of Kraganda, $k = 3$ and $k = 5$.

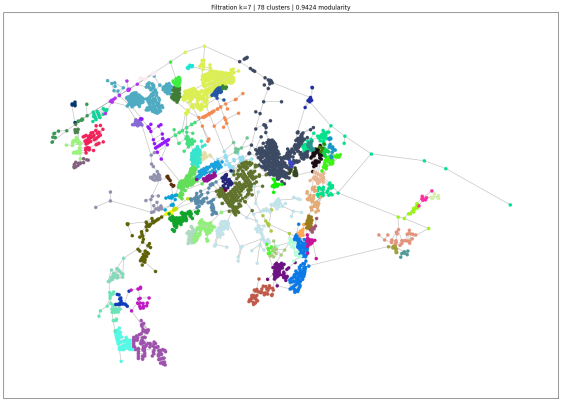


(a) Filtration on knn graph $k = 3$

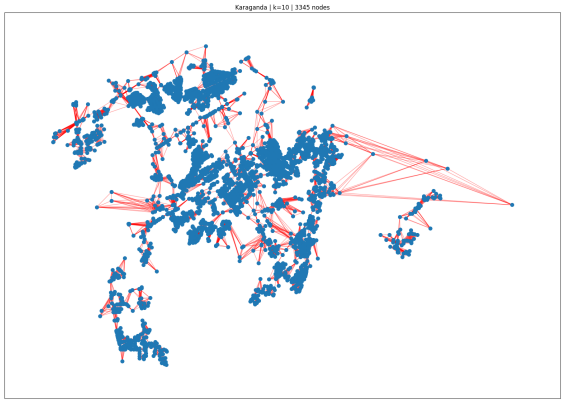


(b) Filtration on knn graph $k = 5$

Figure 7.11: Filtration clustering on knn graphs of Karaganda, $k = 3$ and $k = 5$.

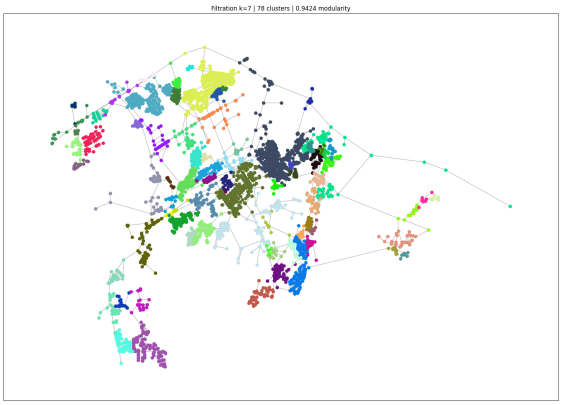


(a) knn graph $k = 7$

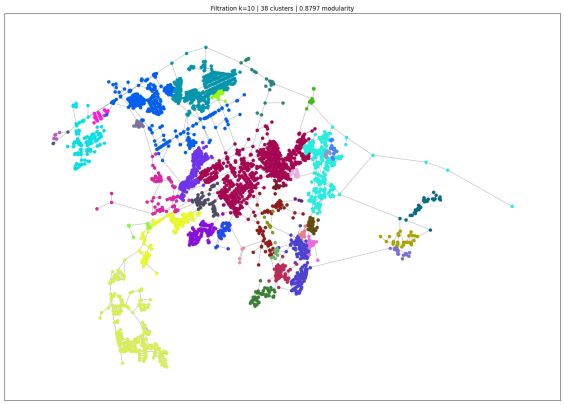


(b) knn graph $k = 10$

Figure 7.12: Knn graphs of Karaganda, $k = 7$ and $k = 10$.

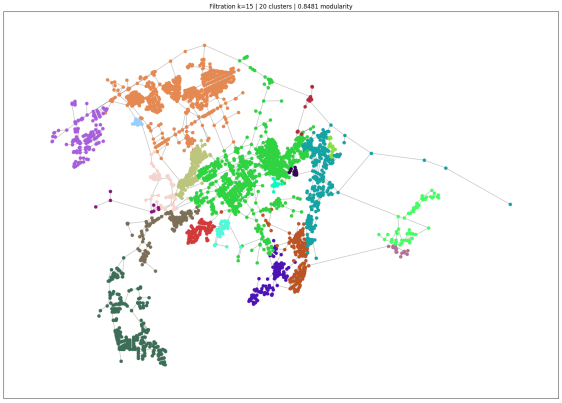


(a) Filtration on knn graph $k = 7$

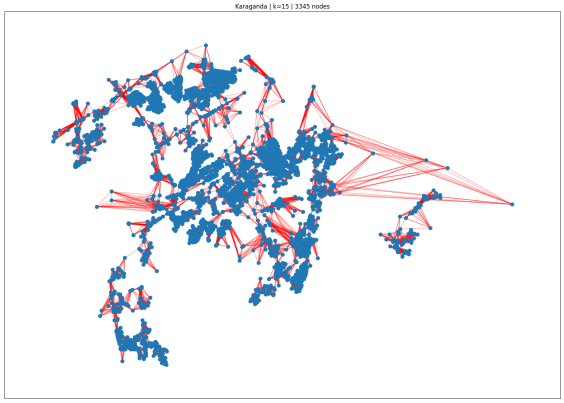


(b) Filtration on knn graph $k = 10$

Figure 7.13: Filtration clustering on knn graphs of Karaganda, $k = 7$ and $k = 10$.



(a) Filtration on knn graph $k = 15$

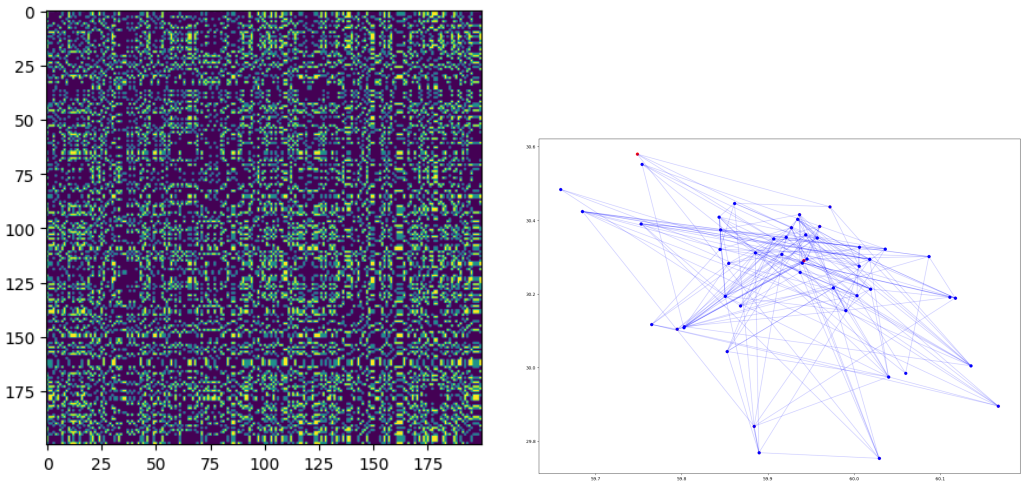


(b) knn graph $k = 15$

Figure 7.14: Knn graph of Karaganda and Filtration clustering on it, $k = 15$.

7.3 Vehicle Routing

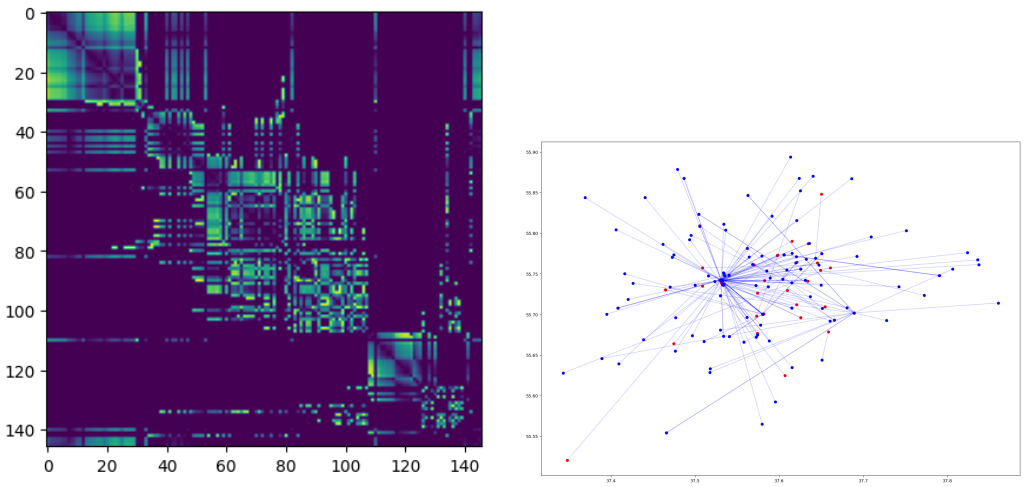
Synthetic data



(a) Time adjacency matrix (b) Addresses

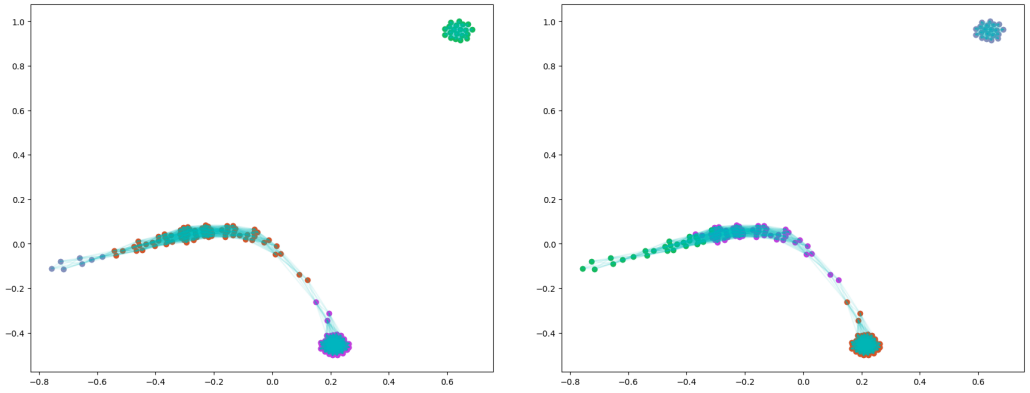
Figure 7.15: Time connectivity between deliveries and metric graph of deliveries: start and end points. It is clear that timestamps were generated with uniform distribution.

Real data



(a) Time adjacency matrix (b) Addresses

Figure 7.16: Time connectivity between deliveries and metric graph of deliveries: start and end points. Real deliveries cluster around specific timestamps (e.g. end of day).



(a) Louvain (b) Leiden

Figure 7.17: Both Leiden and Louvain found 4 clusters, which are different in size.