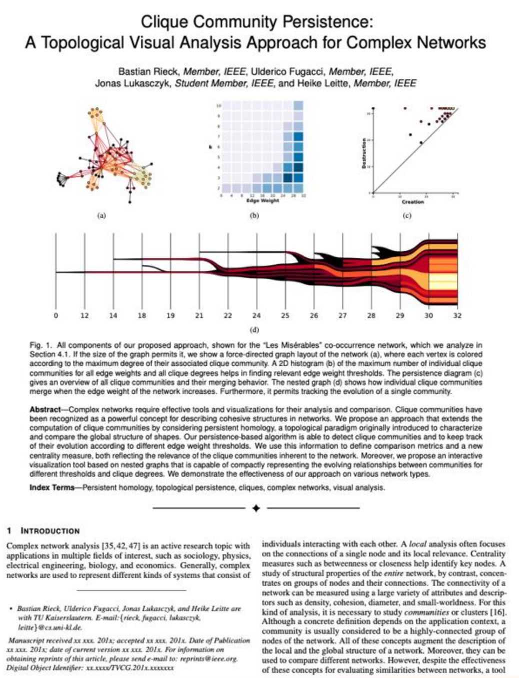
Clique Community Persistence for Complex Networks:

an application to *“Les Miserablés”* co-occurrence network

September, 4th 2024
Fabiola Bertocchi - 248474

Index



Clique Community Persistence:
A Topological Visual Analysis Approach for Complex Networks

Bastian Rieck, *Member, IEEE*, Ulderico Fugacci, *Member, IEEE*,
Jonas Lukasczyk, *Student Member, IEEE*, and Heike Leitte, *Member, IEEE*

(a) (b) (c)

(d)

Fig. 1. All components of our proposed approach, shown for the "Les Misérables" co-occurrence network, which we analyze in Section 4.1. If the size of the graph permits it, we show a force-directed graph layout of the network (a), where each vertex is colored according to the maximum degree of their associated clique community. A 2D histogram (b) of the maximum number of individual clique communities for all edge weights and all clique degrees helps in finding relevant edge weight thresholds. The persistence diagram (c) gives an overview of all clique communities and their merging behavior. The nested graph (d) shows how individual clique communities merge when the edge weight of the network increases. Furthermore, it permits tracking the evolution of a single community.

Abstract—Complex networks require effective tools and visualizations for their analysis and comparison. Clique communities have been recognized as a powerful concept for describing cohesive structures in networks. We propose an approach that extends the computation of clique communities by considering persistent homology, a topological paradigm originally introduced to characterize and compare the global structure of shapes. Our persistence-based algorithm is able to detect clique communities and to keep track of their evolution according to different edge weight thresholds. We use this information to define comparison metrics and a new centrality measure, both reflecting the relevance of the clique communities inherent to the network. Moreover, we propose an interactive visualization tool based on nested graphs that is capable of compactly representing the evolving relationships between communities for different thresholds and clique degrees. We demonstrate the effectiveness of our approach on various network types.

Index Terms—Persistent homology, topological persistence, cliques, complex networks, visual analysis.

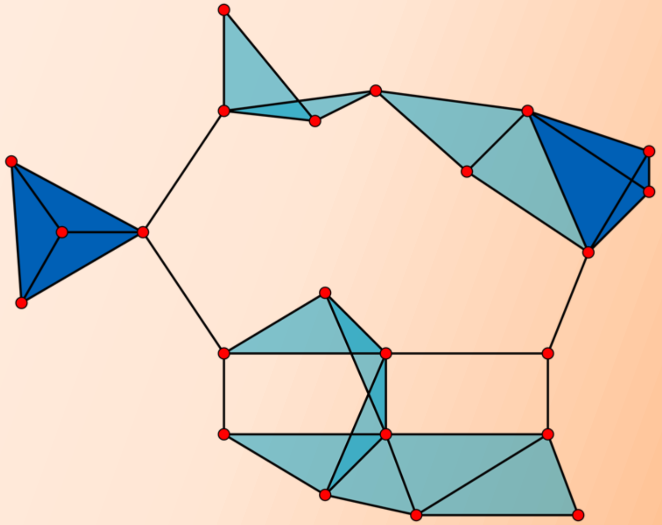
1 INTRODUCTION

Complex network analysis [35, 42, 47] is an active research topic with applications in multiple fields of interest, such as sociology, physics, electrical engineering, biology, and economics. Generally, complex networks are used to represent different kinds of systems that consist of individuals interacting with each other. A *local* analysis often focuses on the connections of a single node and its local relevance. Centrality measures such as betweenness or closeness help identify key nodes. A study of structural properties of the *entire* network, by contrast, concentrates on groups of nodes and their connections. The connectivity of a network can be measured using a large variety of attributes and descriptors such as density, cohesion, diameter, and small-worldness. For this kind of analysis, it is necessary to study *communities* or clusters [16]. Although a concrete definition depends on the application context, a community is usually considered to be a highly-connected group of nodes of the network. All of these concepts augment the description of the local and the global structure of a network. Moreover, they can be used to compare different networks. However, despite the effectiveness of these concepts for evaluating similarities between networks, a tool

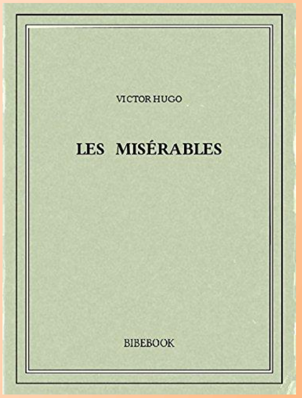
• *Bastian Rieck, Ulderico Fugacci, Jonas Lukasczyk, and Heike Leitte are with TU Kaiserslautern. E-mail:{rieck, fugacci, lukasczyk, leitte}@cs.uni-kl.de.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

- New approach: contents and goals
- k-cliques and k-clique communities
- k-clique connectivity graph
- Homology and persistent homology
- Clique community centrality



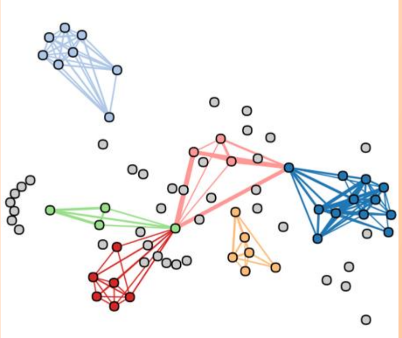
- Case study: *"Les Miserablés"* co-occurrence network



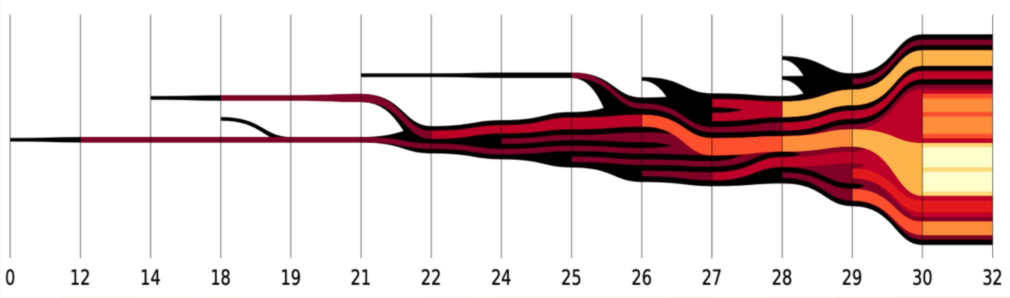
Clique communities and persistence-based method

The big picture:

- Detecting and tracking evolution of clique communities, as both clique degree k and weight threshold w vary → Algorithm based on persistent homology



- Visualizing network structure according to different parameters at the same time → Nested graph



- Comparing different networks → Persistence indicator function and clique community centrality

Complex networks

Complex networks → Systems representing connections between distinct elements

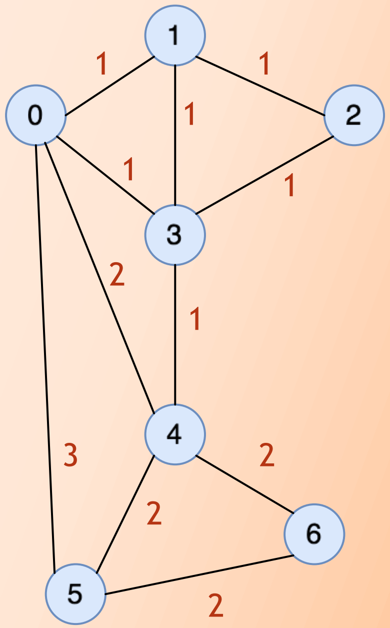
Representation: Weighted graph $\mathcal{G} = (V, E, w)$

where V is the set of vertices,
E is the set of edges $E \subset V \times V$,
w: $V \to \mathbb{R}$ is the weight function

Examples of application fields: sociology, physics, biology, economics, …

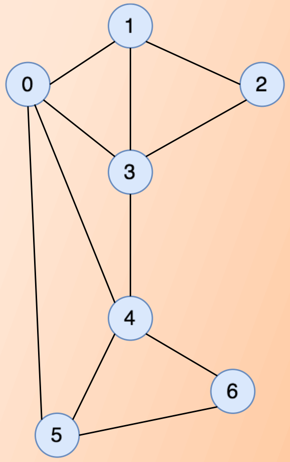
<u>Goal:</u> to study structural properties of the entire network

Toy example:



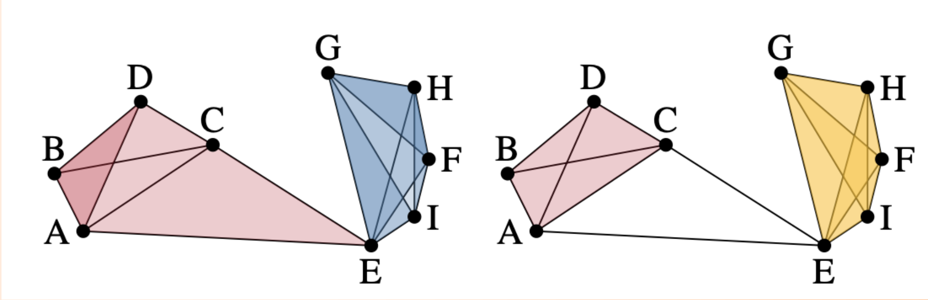
Clique communities

Graph $\mathcal{G} = (V, E)$



- k-clique: complete subgraph of k vertices of $\mathcal{G}$
- k-clique adjacency: two k-cliques are adjacent if they share $k-1$ vertices
- k-clique connectivity: two k-cliques are connected if there exists a sequence of k-cliques of $\mathcal{G}$ s.t. any two consecutive k-cliques are adjacent
- k-clique community: maximal union of k-cliques that are pairwise connected

```
# to extract all cliques from the network G
all_cliques = list(nx.enumerate_all_cliques(G))
```



Filtration of a graph

Filtration of $\mathcal{G}$:

$$\emptyset \subset \mathcal{G}_0 \subset \mathcal{G}_1 \subset \cdots \subset \mathcal{G}_n = \mathcal{G}$$

where the weights of the n vertices are in non-decreasing order $w_1 \leq \cdots \leq w_n$

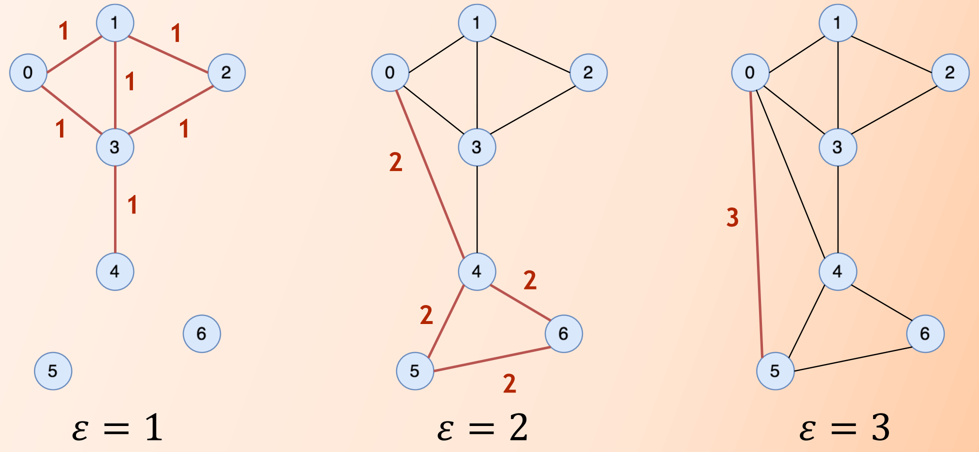
and $\mathcal{G}_i = (V_i, E_i)$ consists of:

$$V_i \coloneqq \{v \in V : w(v) \leq w_i\}$$

$$E_i \coloneqq \{e = \{u, v\} \in E : w(e) \coloneqq \max(w(u), w(v)) \leq w_i\}$$

```
def filtration(G, epsilon):
    """ computes the filtration of given graph, with weight <= epsilon """

    G_epsilon = nx.Graph()
    for (u,v,w) in G.edges(data=True):
        weight = w["weight"] # to extract the weight

        # add edges with weight less or equal than epsilon
        if weight <= epsilon:
            G_epsilon.add_edge(u, v, weight=weight)

    return G_epsilon
```

Toy example:



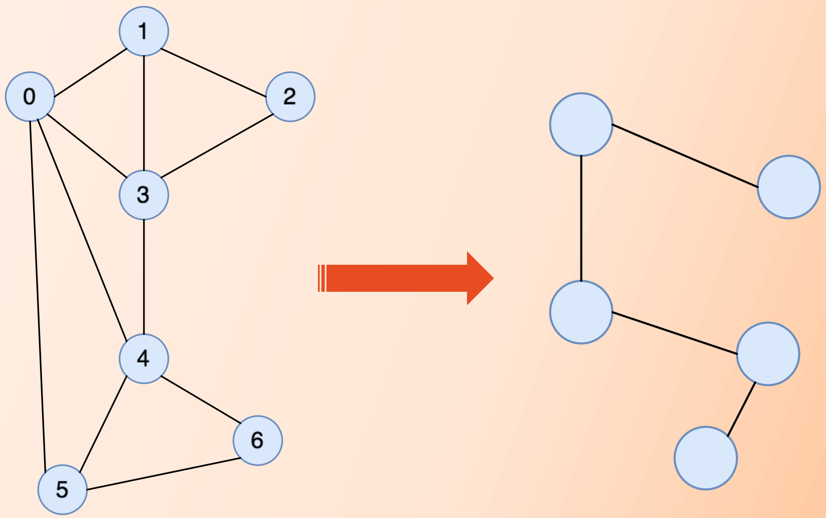
Clique connectivity graph

Weight of a clique σ: $w(\sigma) \coloneqq \max_{\tau \subset \sigma} w(\tau)$

```python
def cliqueWeight(clique, G):
    """ computes the weight of a single k-clique
    as the maximum weight of its subsets """
    clique = list(clique)
    weight = -math.inf

    if len(clique) == 1:     # vertex
        weight = 0

    elif len(clique) == 2:  # edge
        u = clique[0]
        v = clique[1]
        weight = G[u][v]["weight"]

    elif len(clique) >= 3:  # triangle and higher dimensional cliques
        for u in clique:
            for v in clique:
                if u != v:
                    w = G[u][v]["weight"]
                    if weight < w:
                        weight = w

    return weight
```

Clique connectivity graph

Weight of a clique σ: $\quad w(\sigma) \coloneqq \max_{\tau \subset \sigma} w(\tau)$



k-clique connectivity graph $\mathcal{G}^k = (V^k, E^k)$:

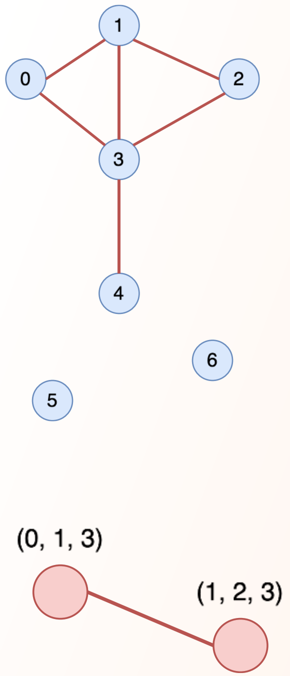
vertices $\longrightarrow$ k-cliques of $\mathcal{G}$

edges $\longrightarrow$ $E^k \coloneqq \left\{ \{\sigma, \sigma'\} \in V^k \times V^k \colon \sigma, \sigma' \text{ are adjacent} \right\}$

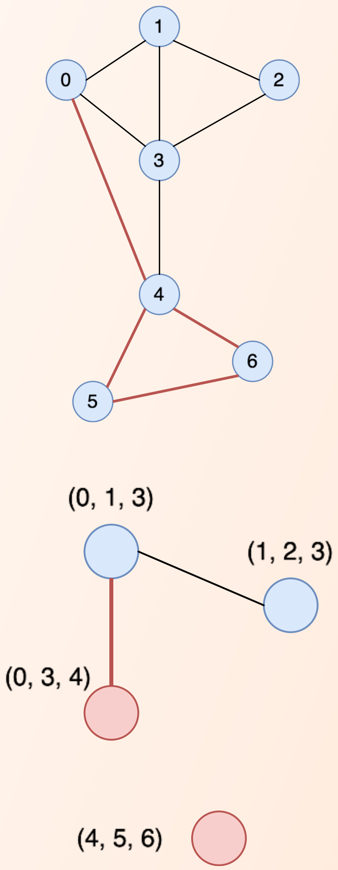
weight $\longrightarrow$ $w(\sigma, \sigma') \coloneqq \max\left(w(\sigma), w(\sigma')\right)$

Clique connectivity graph

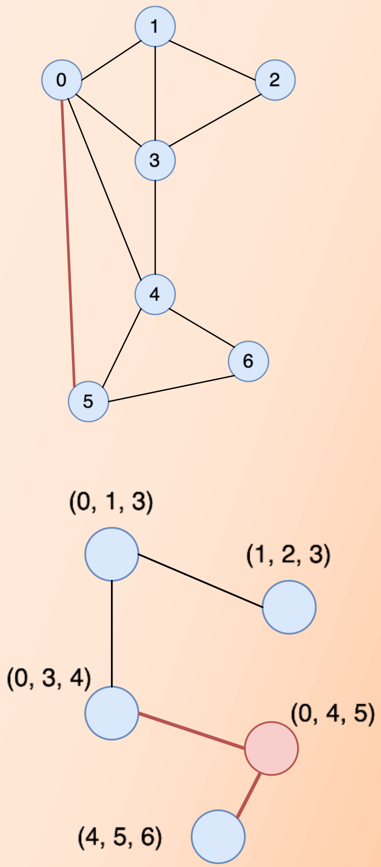
Evolution of 3-clique connectivity graph from toy example:



$\varepsilon = 1$



$\varepsilon = 2$



$\varepsilon = 3$

Clique connectivity graph

```
def connectivityGraph(G, k):
    """ constructs k-clique connectivity graph of G """
    graphs = []
    # to find the maximum weight contained in the given network
    max_weight = max(w['weight'] for u, v, w in G.edges(data=True))

    for w in range(max_weight+1):    # iterating over all weights
        clique_conn_graph = nx.Graph()
        G_w = filtration(G, w)               # compute the w-th filtration
        cliques_w = list(nx.enumerate_all_cliques(G_w))   # to extract all cliques of G_w

        for c1 in cliques_w:
            if len(c1) == k: # consider only k-cliques
                c1 = set(c1)
                if tuple(c1) not in clique_conn_graph.nodes():
                    clique_conn_graph.add_node(tuple(c1))   # add a node for every k-clique of G_w
                for c2 in cliques_w:
                    if len(c2) == k:    # consider only k-cliques
                        c2 = set(c2)
                        if c1 == c2:
                            continue

                        # to check whether the two considered cliques are adjacent (whether they share k-1 vertices)
                        diff = 0
                        for element in c1:
                            if element not in c2:
                                diff += 1
                        if diff == 1:   # the two cliques share k-1 vertices, hence they are adjacent
                            clique_conn_graph.add_edge(tuple(c1), tuple(c2), weight=weightConnEdges(c1, c2, G_w))
        graphs.append(clique_conn_graph)
    return clique_conn_graph, graphs
```

Homology and persistent homology

- Boundary map: $\begin{aligned} \partial_n \colon C_n(K) &\to C_{n-1}(K) \\ c &\mapsto \partial c \end{aligned}$ with K a simplicial complex
  $C_n(K)$ the set of n-chains of K

- n-th simplicial homology group: $H_n(K) = \dfrac{Z_n(K)}{B_n(K)}$ where $Z_n(K) = \ker(\partial_n)$ is the vector space of n-cycles
  $B_n(K) = \mathrm{imm}(\partial_{n+1})$ is the vector space of n-boundaries

  reveals the presence of n-dimensional holes in the complex

- Persistent homology: describes the changes in homology as an object evolves with respect to a parameter

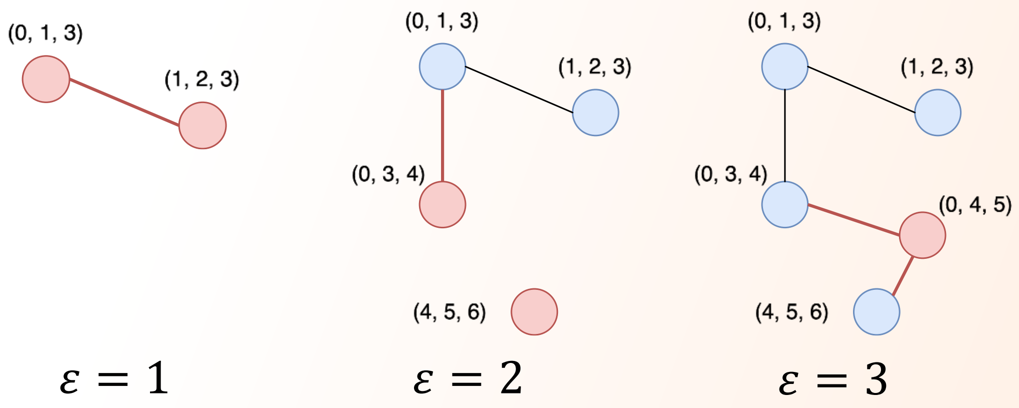
  Persistence pair of a homological class: $(c, d) \in \mathbb{R}^2$

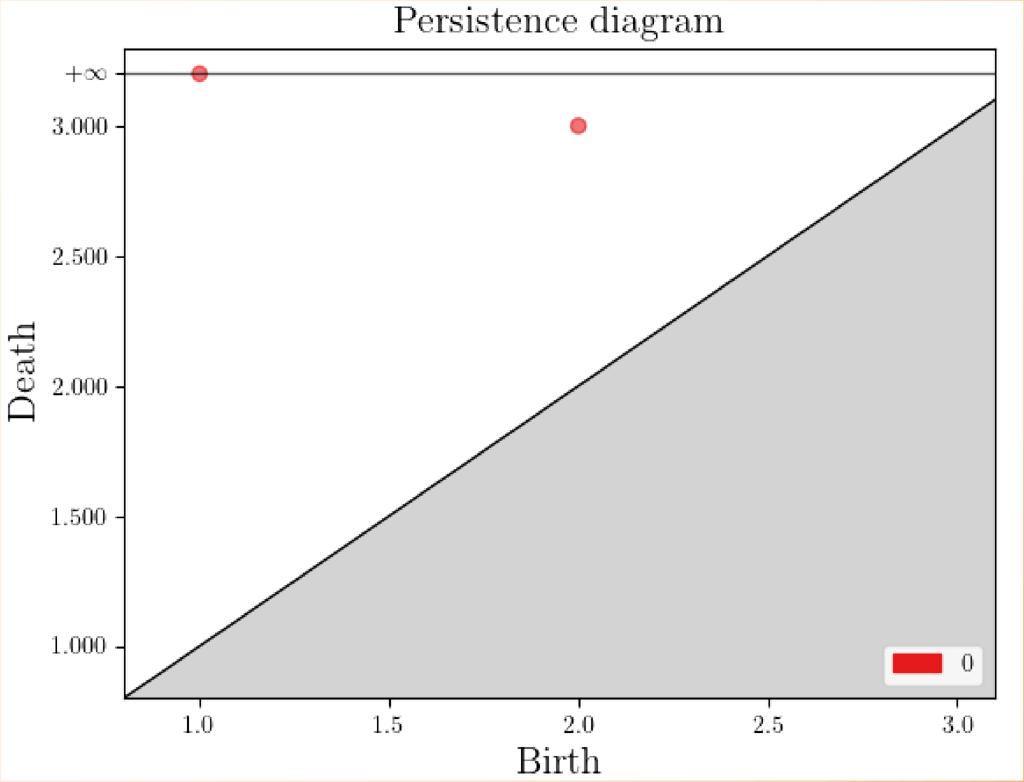
  where c is the birth of the homological class,
  d is the death of the homological class

  Persistence: $\mathrm{pers}(c, d) := |d - c|$

Persistent homology for clique communities

Toy example:





Persistence pairs:
$(2,3)$, $(1, +\infty)$

Persistent homology for clique communities

Algorithm 1 0-dimensional persistent homology calculation

Require: A weighted graph $\mathcal{G}$

$\texttt{UF} \leftarrow \emptyset$ ▷ Initialize an empty Union–Find structure
$\mathcal{D} \leftarrow \emptyset$ ▷ Initialize an empty persistence diagram
for every edge $(u,v) \in \mathcal{G}$ in ascending order of its weight **do**
  $c \leftarrow \texttt{UF.Find}(u)$
  $c' \leftarrow \texttt{UF.Find}(v)$
  if $\mathrm{w}(c) < \mathrm{w}(c')$ **then** ▷ c is the older component; merge c' into it
    $\texttt{UF.Union}(c', c)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \left(\mathrm{w}(c'), \mathrm{w}(u,v)\right)$
  else ▷ c' is the older component; merge c into it
    $\texttt{UF.Union}(c, c')$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \left(\mathrm{w}(c), \mathrm{w}(u,v)\right)$
  end if
end for
return $\mathcal{D}$

```python
def cliquePersistentHomology(graphs, G_old):
    """ computes 0-dimensional persistent homology of k-clique connectivity graph """
    uf = UnionFind()    # initialize empty Union-Find structure
    d = []              # initialize empty persistence diagram
    for w in range(len(graphs)):
        elements = extractEdgesWithWeight(graphs[w])
        for node in graphs[w].nodes():
            uf.add(node)
        for (u, v, weight) in sorted(elements, key=lambda x: (x[2], x[0], x[1])):
            c1 = uf.find(u)
            c2 = uf.find(v)
            if c1 == c2:
                continue
            w1 = cliqueWeight(uf[c1], G_old)
            w2 = cliqueWeight(uf[c2], G_old)
            if w1 <= w2:    # c1 is older component, merge c2 into it
                uf.union(uf[c1], uf[c2])
                if w2 != weight:
                    d.append((0, (w2, weight)))
            else:           # c2 is older component, merge c1 into it
                uf.union(uf[c2], uf[c1])
                if w1 != weight:
                    d.append((0, (w1, weight)))
    # to add persistence of components that are never destroyed
    for comp in uf.components():
        weight = []
        for elem in comp:
            weight.append(cliqueWeight(elem, G_old))
        birth = min(weight)
        d.append((0, (birth, math.inf)))
    return d
```

Persistence indicator function

Persistence indicator function of a persistence diagram $\mathcal{D}$:

$$\mathbb{I}_{\mathcal{D}}\colon \mathbb{R} \to \mathbb{N}$$
$$\varepsilon \longmapsto \text{card}\,\{(c,d) \in \mathcal{D} \mid \varepsilon \in (c,d)\}$$

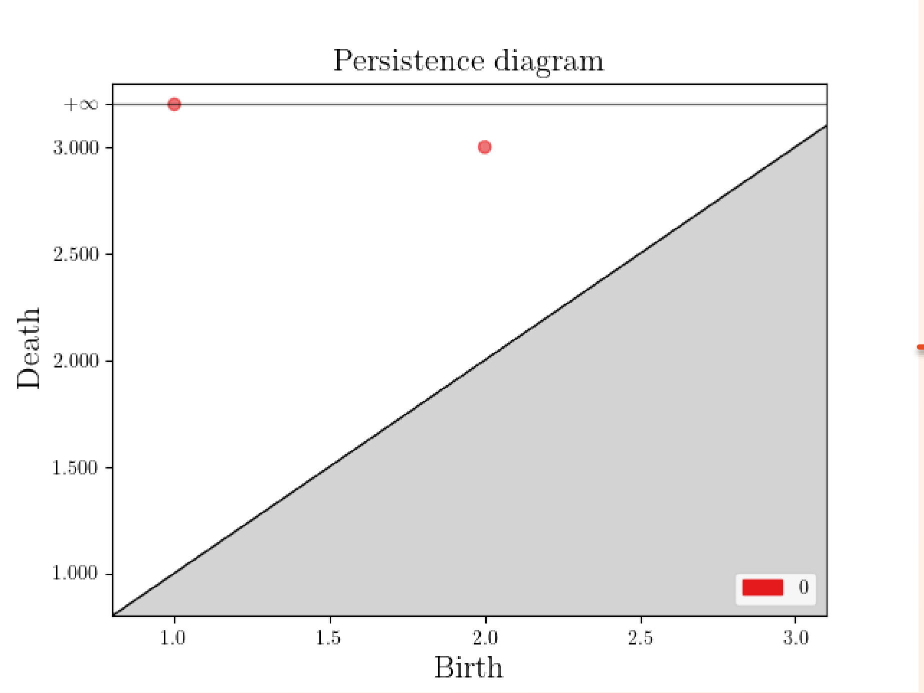
number of active connected components at threshold ε

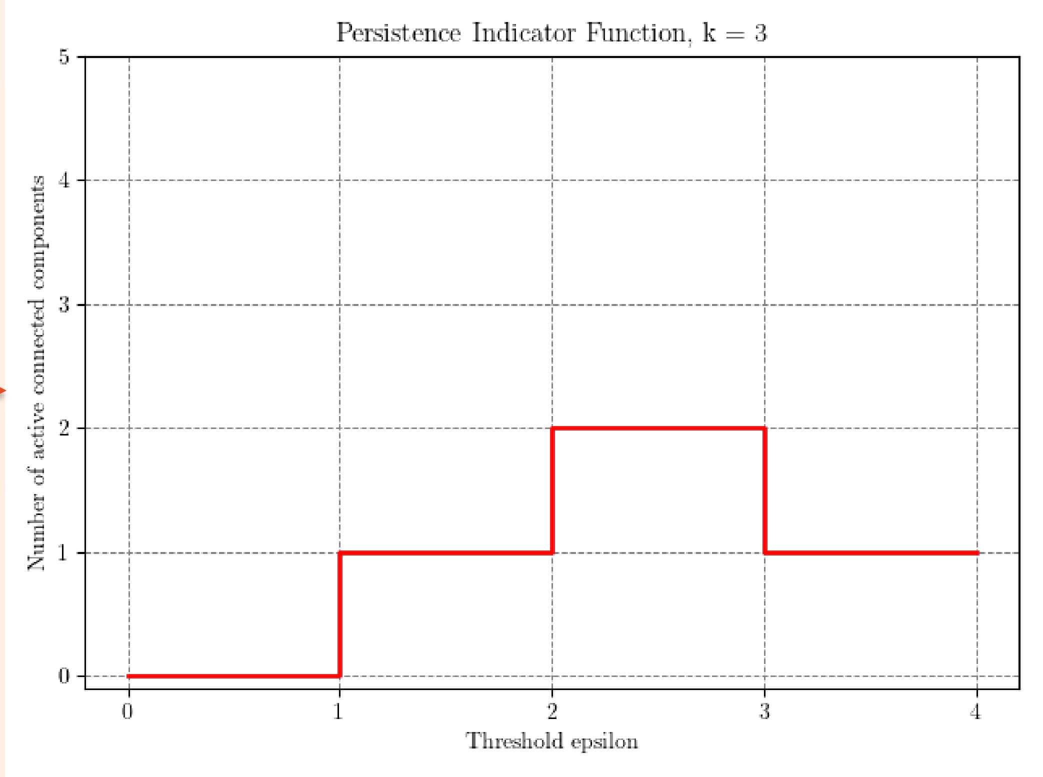
```
def persistenceIndicator(diagram, epsilon):
    """ computes number of active connected components for given threshold epsilon """

    n = 0
    for comp in diagram:
        birth = comp[1][0]
        death = comp[1][1]
        if epsilon >= birth and epsilon < death:
            n += 1

    return n
```

L^p distance: $\quad \text{dist}(\mathbb{I}_{\mathcal{D}_1}, \mathbb{I}_{\mathcal{D}_2}) := \left(\int \left|\mathbb{I}_{\mathcal{D}_1}(x) - \mathbb{I}_{\mathcal{D}_2}(x)\right|^p dx\right)^{1/p}$

Persistence indicator function

Toy example:





Clique community centrality

Analysis of importance of given node

Clique community centrality

$$\Gamma_c(v) \coloneqq \sum_{v \in C} pers(C)$$

where C indicates all clique communities the vertex is part of

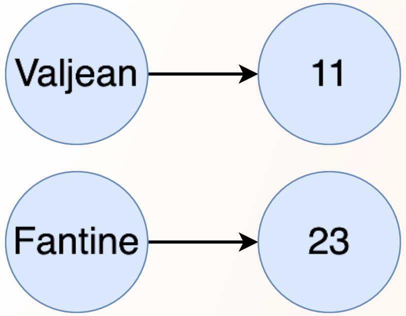
Persistence ⟷ Relevance

```
def cliqueCommunityCentrality(v, G, weight_infinity=32):
    """ measures the relevance of vertex v considering persistence of all communities v is part of """
    all_cliques = list(nx.enumerate_all_cliques(G))
    len_max_clique = len(max(all_cliques, key=len))
    centrality = 0
    ph = []

    # to compute persistence of all communities for all k
    for k in range(2, len_max_clique+1):
        clique_conn_graph, graphs = connectivityGraph(G, k)
        ph.append(computePersistence(graphs, G))

    for element in ph:
        for el in element:
            for e in el[0]:
                if v in e:  # check whether v is in the considered community
                    birth = el[1][0]
                    death = el[1][1]

                    #  extract persistence of the community
                    persistence = abs(death-birth) if death != float('inf') else weight_infinity
                    centrality += persistence
                    break

    return centrality
```

Case study: “*Les Miserablés*” network

Data set:

- Nodes: novel’s characters



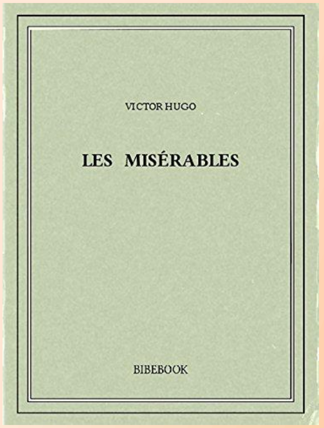
- Edge weights: co-occurrences between characters

↓

invert weights to have:

Weights ⟷ Proximity

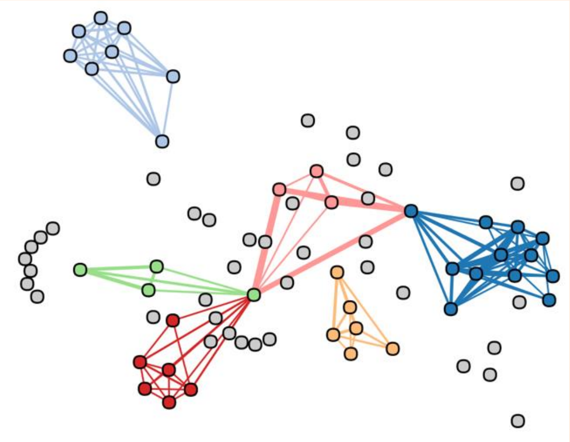
```
for (u,v,w) in G_old.edges(data=True):
    weight = max_weight - w["weight"]
    G[u][v]["weight"] = weight
```



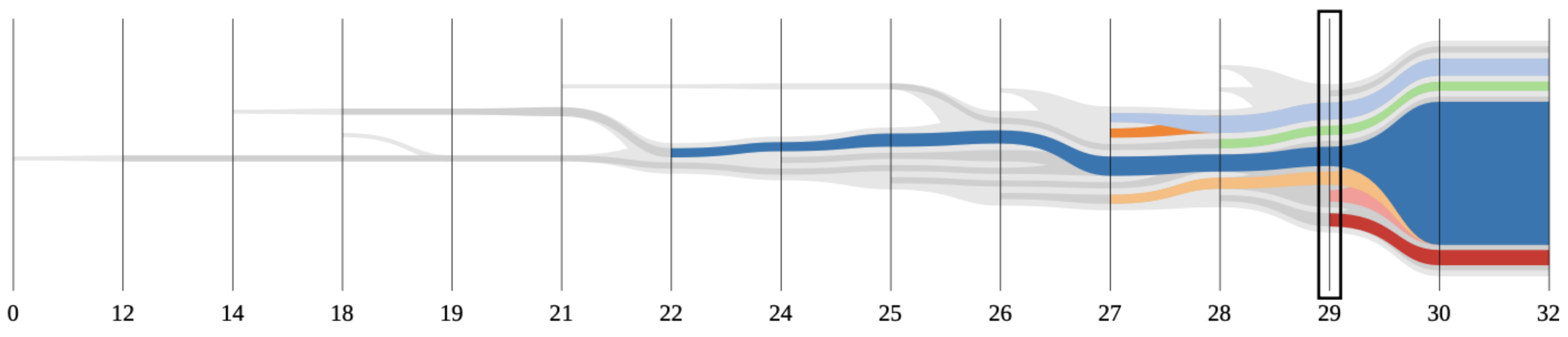
Case study: *"Les Miserablés"* network

Cliques: up to $k = 10$

```
# to extract all cliques of G
all_cliques = list(nx.enumerate_all_cliques(G))

# length of clique with maximal length
len_max_clique = len(max(all_cliques, key=len))
```

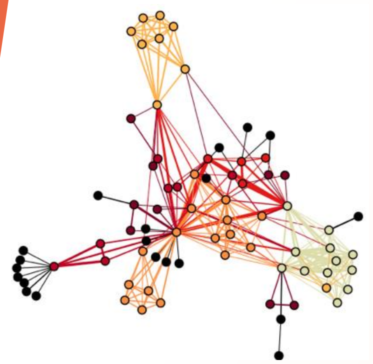


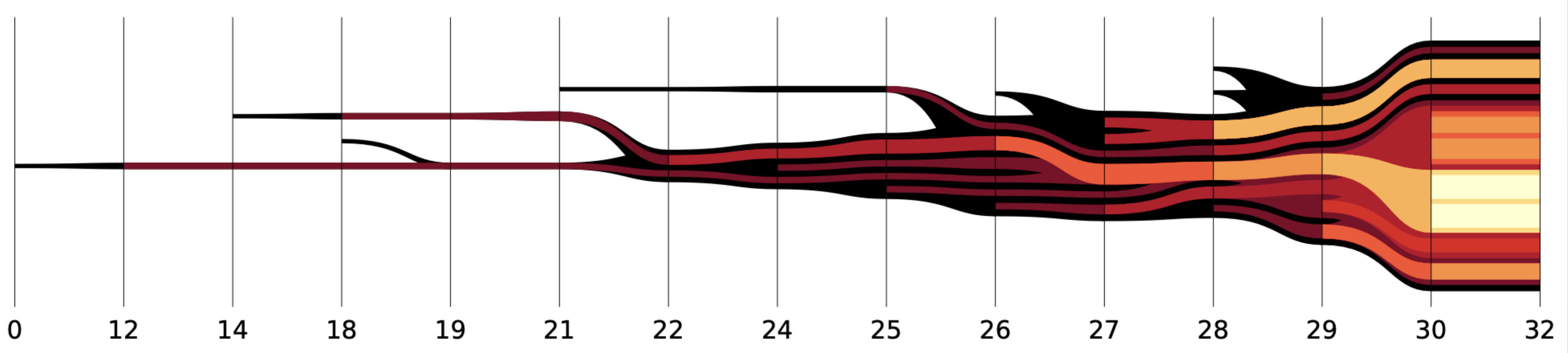
For $k = 4$ and $w = 29$: 6 communities ⟶ significant groups of characters



Case study: *“Les Miserablés”* network

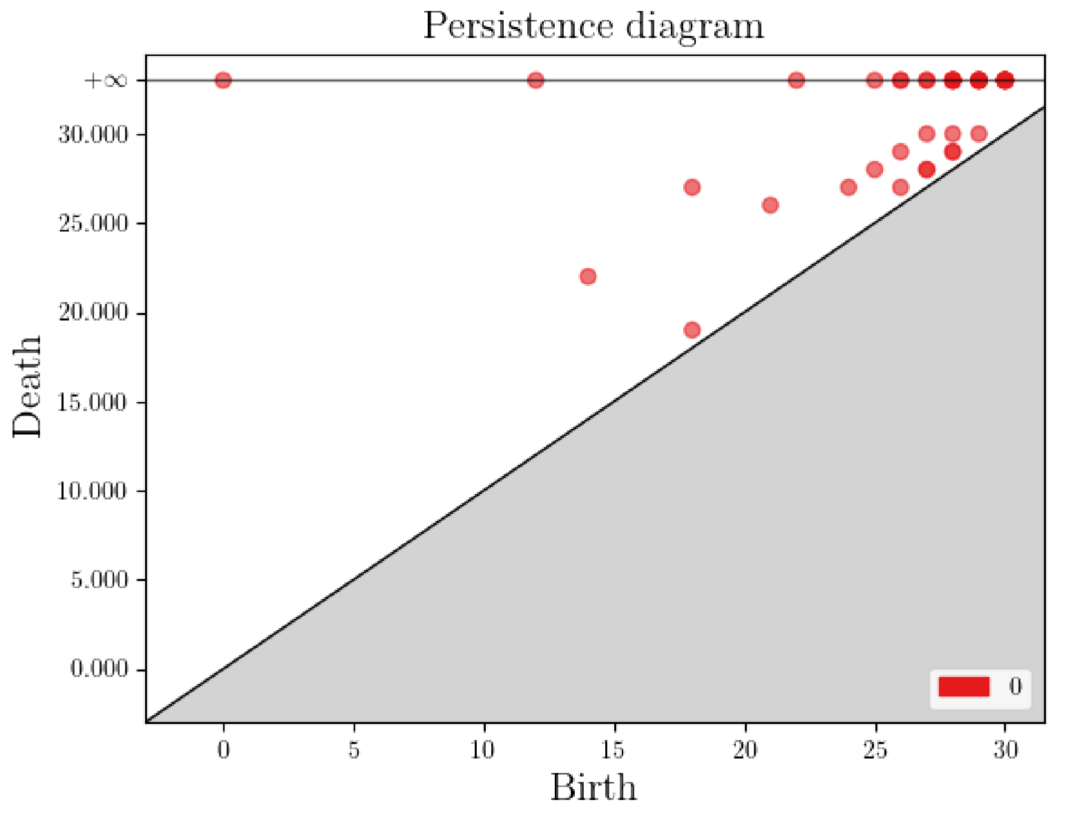
Visualization: nested graph





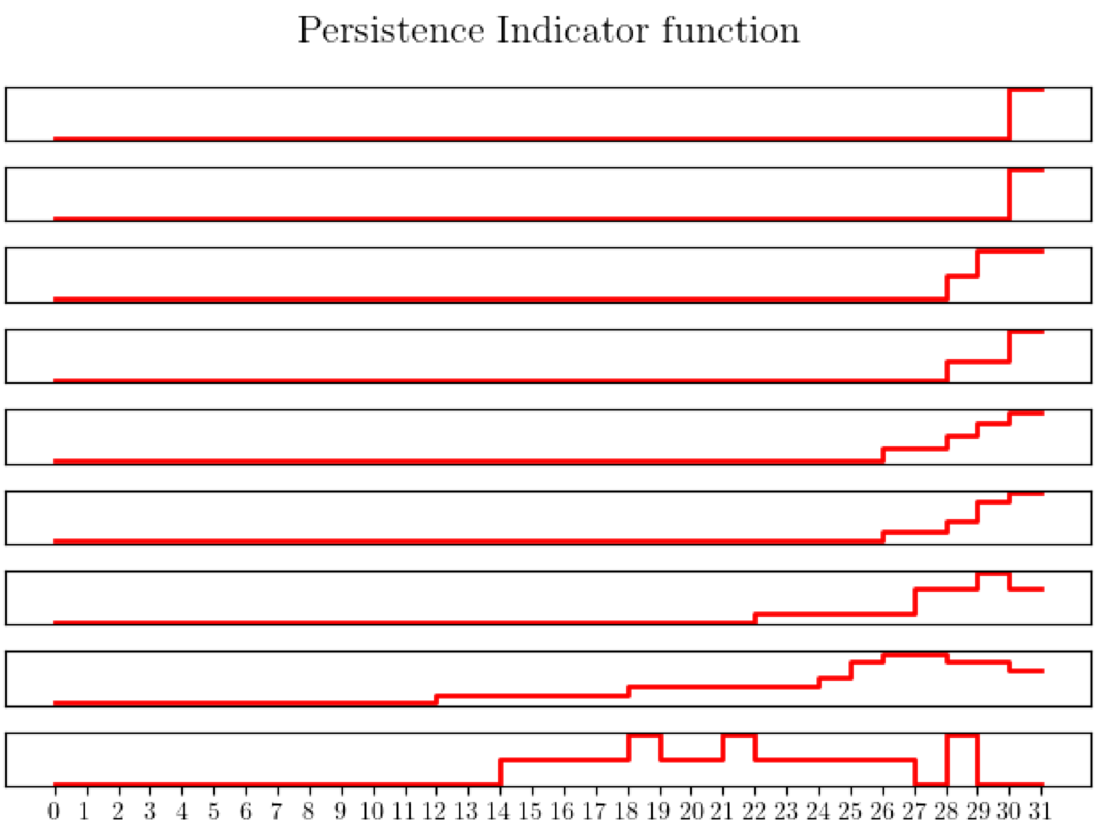
Case study: “*Les Miserablés*” network

Persistence diagram:



Case study: “*Les Miserablés*” network

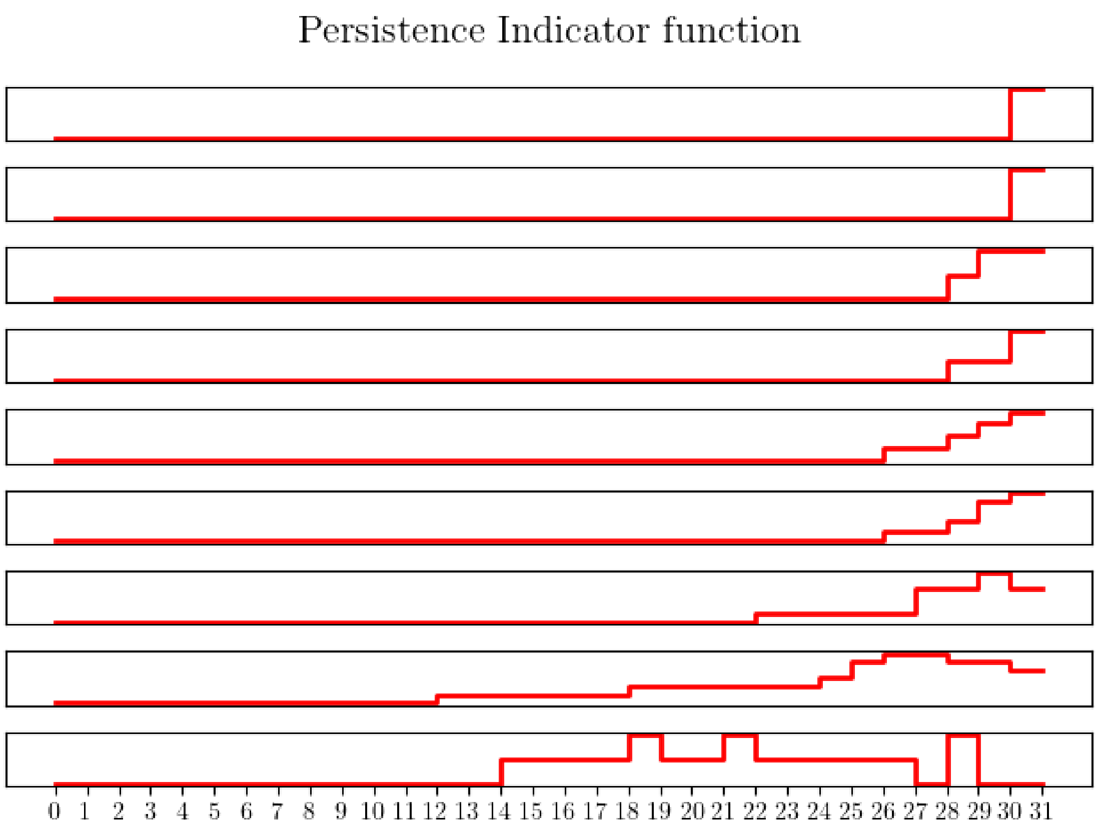
Persistence indicator function:

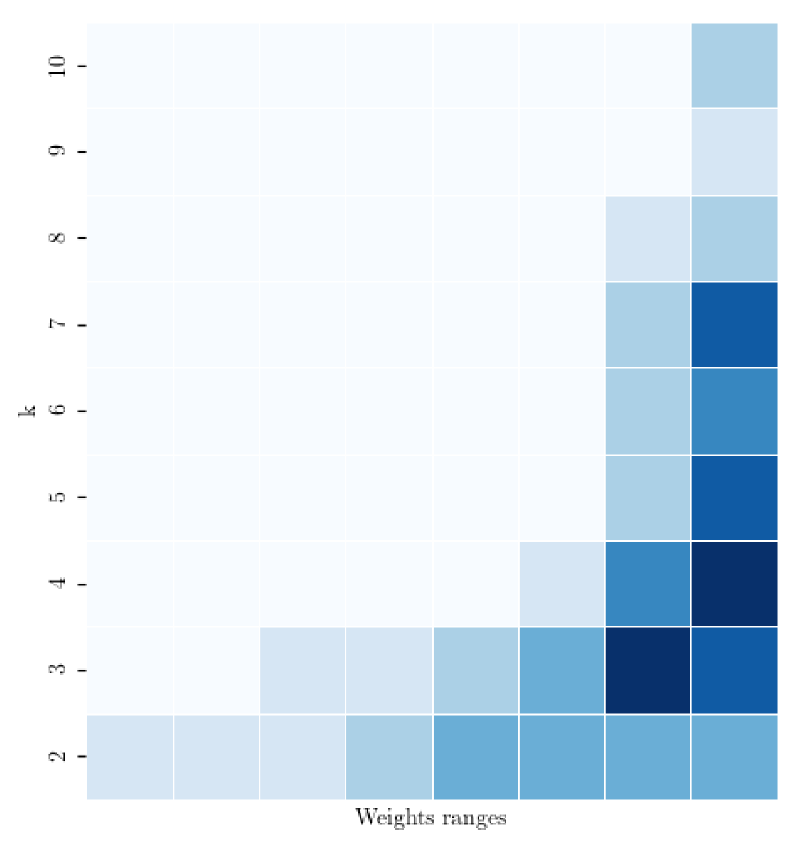



```python
def glyph(diag, divider, length):
    """ condensed glyph that gives a summary of clique community activity """

    steps = length // divider   # discretize the domain into uniformly spaced bins
    values = []
    for step in range(steps):
        value = -math.inf
        for epsilon in range(step * divider, step * divider + divider+1):
            # maximum value of persistence indicator in that range
            value = max(value, persistenceIndicator(diag, epsilon))
        values.append(value)
    return values
```

Case study: “*Les Miserablés*” network

Persistence indicator function:





Case study: “*Les Miserablés*” network

Clique community centrality:

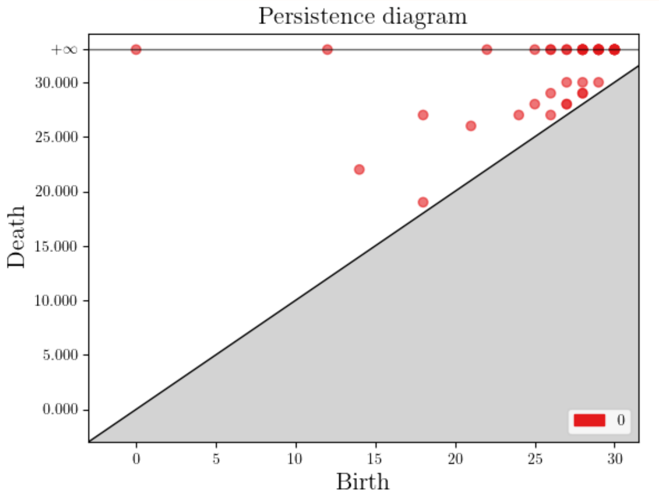
BC	CC	EC	CCC
Valjean	Valjean	Gavroche	Valjean
Myriel	Marius	Valjean	Gavroche
Gavroche	Javert	Enjolras	Fantine
Marius	Thénardier	Marius	Marius
Fantine	Gavroche	Bossuet	Enjolras

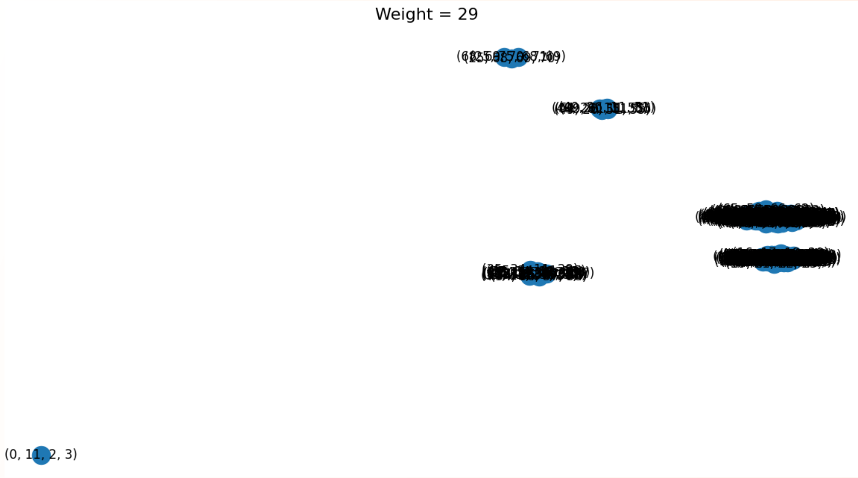
Legend:
- Valjean ↔ 11
- Gavroche ↔ 48
- Fantine ↔ 23
- Marius ↔ 55
- Enjolras ↔ 58

```
Node with highest centrality measure: 11 value: 489
All (nodes, values) ordered from highest centrality measure to lowest:
 [(11, 489), (48, 477), (23, 362), (55, 361), (58, 357), (64, 356), (6
2, 356), (59, 356), (65, 348), (63, 348), (61, 348), (66, 301), (60, 3
01), (57, 295), (25, 265), (16, 261), (69, 241), (68, 241), (71, 235),
 (70, 235), (27, 233), (24, 233), (29, 232), (22, 229), (21, 229), (20
, 229), (19, 229), (18, 229), (17, 229), (76, 227), (41, 204), (75, 20
3), (26, 201), (38, 196), (37, 196), (36, 196), (35, 196), (34, 196),
(49, 157), (51, 156), (54, 130), (42, 107), (31, 107), (3, 104), (2, 1
04), (0, 104), (72, 102), (43, 102), (28, 72), (12, 68), (74, 66), (73
, 66), (44, 66), (33, 66), (30, 66), (56, 64), (39, 64), (47, 34), (45
, 34), (8, 34), (67, 33), (53, 32), (52, 32), (50, 32), (46, 32), (40,
 32), (32, 32), (15, 32), (14, 32), (13, 32), (10, 32), (9, 32), (7, 3
2), (6, 32), (5, 32), (4, 32), (1, 32)]
First five nodes with highest centrality measure: [(11, 489), (48, 477
), (23, 362), (55, 361), (58, 357)]
```

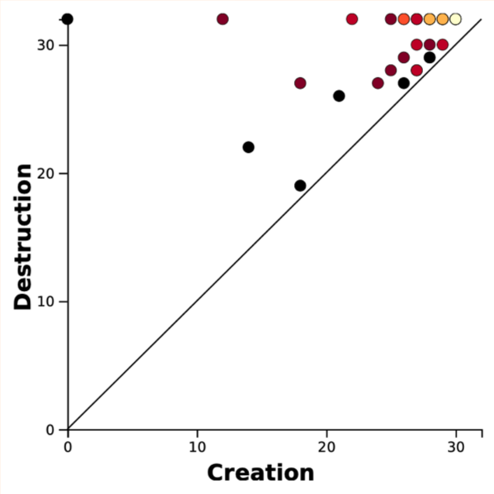
Comparison of results

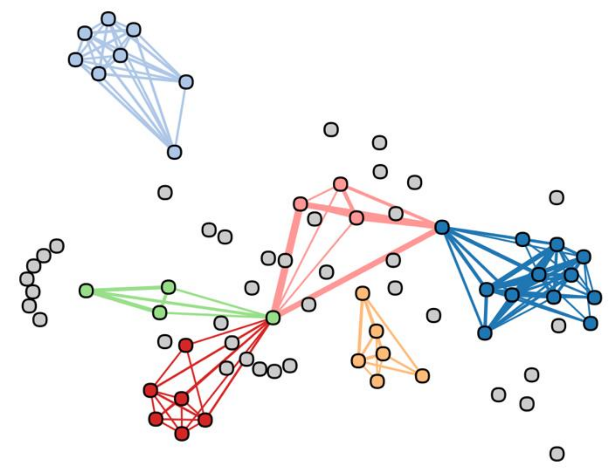
My results





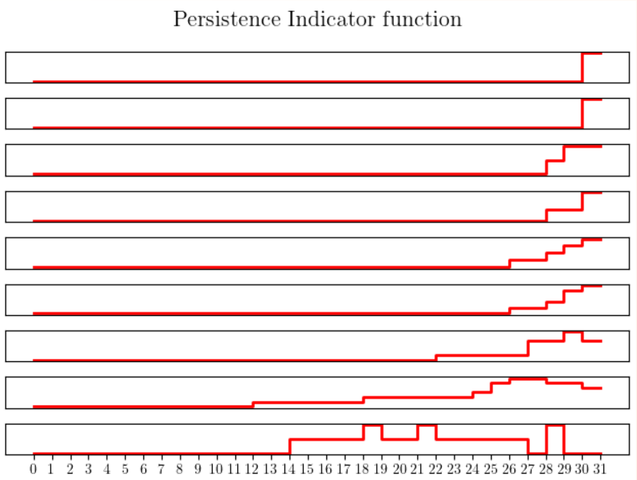
Results from the paper

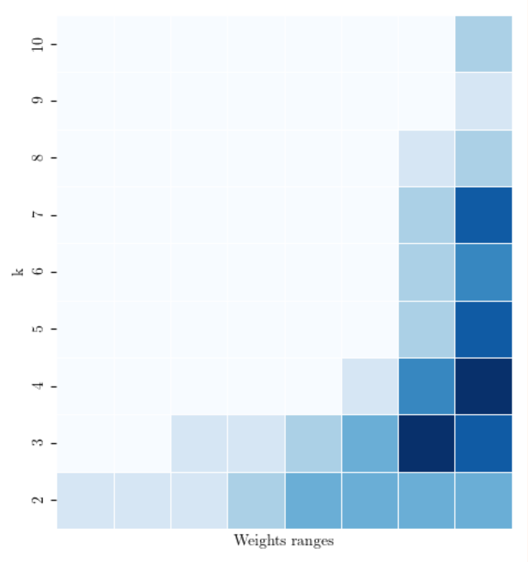




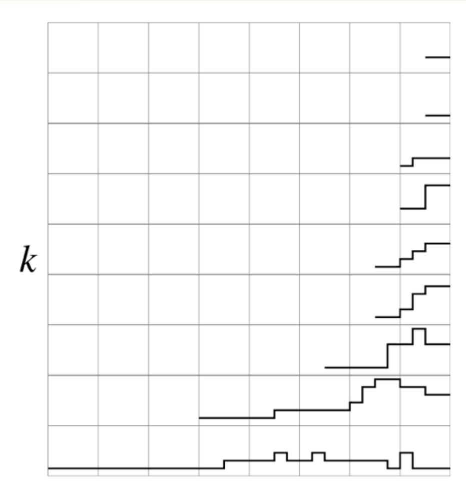
Comparison of results

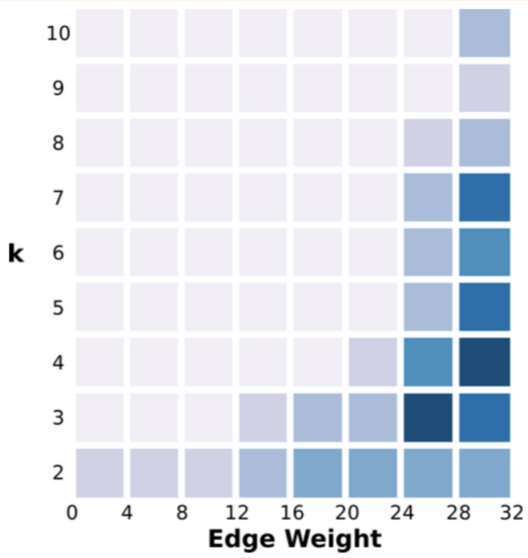
My results





Results from the paper





References

[1] "*Clique Community Persistence: A Topological Visual Analysis Approach for Complex Networks*", B. Rieck, U. Fugacci, J. Lukasczyk and H. Leitte, in *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 822-831, Jan. 2018, doi: 10.1109/TVCG.2017.2744321

Union-Find data structure:

[2] Adaptation of https://github.com/deehzee/unionfind

My project implementation can be found on the following GitHub repository:

https://github.com/fabertocchi/clique-community-persistence

THANKS FOR YOUR ATTENTION!