# Methods of Network Analysis

# Clustering and Blockmodeling

## 3. Graphs and Clustering

Vladimir Batagelj
University of Ljubljana, Slovenia

University of Konstanz, Algorithms and Data Structures

June 10, 2002, 10-12h, room F 426



## Clustering and Networks

- clustering with relational constraint
- transforming data into graphs (neighbors)
- clustering of networks; dissimilarities between graphs (networks)
- clustering of vertices / links; dissimilarities between vertices
- clustering in large networks





## Clustering with relational constraint

Suppose that the units are described by attribute data $a \colon \mathbf{U} \to [\mathbf{U}]$ and related by a binary *relation* $R \subseteq \mathbf{U} \times \mathbf{U}$ that determine the *relational data* $(\mathbf{U}, R, a)$.

We want to cluster the units according to the similarity of their descriptions, but also considering the relation $R$ – it imposes constraints on the set of feasible clusterings, usually in the following form:

$$\mathbf{\Phi}(R) = \{\mathbf{C} \in P(\mathbf{U}) : \text{each cluster } C \in \mathbf{C} \text{ is a subgraph } (C, R \cap C \times C) \text{ in the graph } (\mathbf{U}, R) \text{ of the required type of connectedness}\}$$





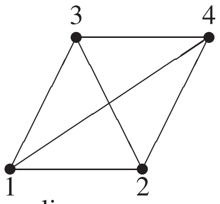
### Some types of relational constraints

We can define different types of sets of feasible clusterings for the same relation $R$. Some examples of *types of relational constraint* $\Phi^i(R)$ are

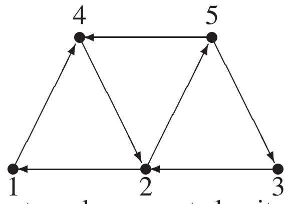
| type of clusterings | type of connectedness |
|---|---|
| $\Phi^1(R)$ | weakly connected units |
| $\Phi^2(R)$ | weakly connected units that contain at most one center |
| $\Phi^3(R)$ | strongly connected units |
| $\Phi^4(R)$ | clique |
| $\Phi^5(R)$ | the existence of a trail containing all the units of the cluster |

A set of units $L \subseteq C$ is a *center* of cluster $C$ in the clustering of type $\Phi^2(R)$ iff the subgraph induced by $L$ is strongly connected and $R(L) \cap (C \setminus L) = \emptyset$.

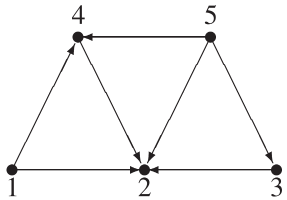
## Some graphs of different types



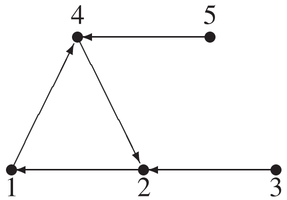

a clique




strongly connected units




weakly connected units




weakly connected units with a center $\{1, 2, 4\}$





## Properties of relational constraints

The sets of feasible clusterings $\Phi^i(R)$ are linked as follows:

$\Phi^4(R) \subseteq \Phi^3(R) \subseteq \Phi^2(R) \subseteq \Phi^1(R)$

$\Phi^4(R) \subseteq \Phi^5(R) \subseteq \Phi^2(R)$

If the relation $R$ is symmetric, then $\Phi^3(R) = \Phi^1(R)$

If the relation $R$ is an equivalence relation, then $\Phi^4(R) = \Phi^1(R)$

Here are also examples of the corresponding fusibility predicates:

$\psi^1(C_1, C_2) \equiv \exists \mathrm{X} \in C_1 \exists \mathrm{Y} \in C_2 : (\mathrm{X}R\mathrm{Y} \vee \mathrm{Y}R\mathrm{X})$

$\psi^2(C_1, C_2) \equiv (\exists \mathrm{X} \in L_1 \exists \mathrm{Y} \in C_2 : \mathrm{X}R\mathrm{Y}) \vee (\exists \mathrm{X} \in C_1 \exists \mathrm{Y} \in L_2 : \mathrm{Y}R\mathrm{X})$

$\psi^3(C_1, C_2) \equiv (\exists \mathrm{X} \in C_1 \exists \mathrm{Y} \in C_2 : \mathrm{X}R\mathrm{Y}) \wedge (\exists \mathrm{X} \in C_1 \exists \mathrm{Y} \in C_2 : \mathrm{Y}R\mathrm{X})$

$\psi^4(C_1, C_2) \equiv \forall \mathrm{X} \in C_1 \forall \mathrm{Y} \in C_2 : (\mathrm{X}R\mathrm{Y} \wedge \mathrm{Y}R\mathrm{X})$

For $\psi^3$ the property F5 fails.

We can use both hierarchical and local optimization methods for solving some types of problems with relational constraint (Ferligoj, Batagelj 1983).





# Neighborhood Graphs

For a given dissimilarity $d$ on the set of units $\mathbf{U}$ we can define several graphs:

The *k nearest neighbors graph* $\mathbf{G}_N(k) = (\mathbf{U}, A)$

$$(\mathrm{X}, \mathrm{Y}) \in A \Leftrightarrow \mathrm{Y} \text{ is among the } k \text{ closest neighbors of } \mathrm{X}$$

By setting for $a(\mathrm{X}, \mathrm{Y}) \in A$ its value to $w(a) = d(\mathrm{X}, \mathrm{Y})$ we obtain a network.
In the case of equidistant pairs of units we have to decide – or to include them all in the graph, or specify an additional selection rule.

A special case of the $k$ nearest neighbors graph is the *nearest neighbor graph* $\mathbf{G}_N(1)$. We shall denote by $\mathbf{G}^*_{NN}$ the graph with included all equidistant pairs, and by $\mathbf{G}_{NN}$ a graph where a single nearest neighbor is always selected.

The *fixed-radius neighbors graph* $\mathbf{G}_B(r) = (\mathbf{U}, E)$

$$(\mathrm{X} : \mathrm{Y}) \in E \Leftrightarrow d(\mathrm{X}, \mathrm{Y}) \leq r$$

There are several papers on efficient algorithms for determining the neighborhood graphs (Fukunaga, Narendra (1975), Dickerson, Eppstein (1996), Chávez & (1999), Murtagh (1999)). These graphs are a bridge between data and network analysis.





## Structure and properties of the nearest neighbor graphs

Let $\mathbf{N} = (\mathbf{U}, A, w)$ be a nearest neighbor network. A pair of units $\mathrm{X}, \mathrm{Y} \in \mathbf{U}$ are *reciprocal nearest neighbors* or RNNs iff $(\mathrm{X}, \mathrm{Y}) \in A$ and $(\mathrm{Y}, \mathrm{X}) \in A$.

Suppose $\mathrm{card}(\mathbf{U}) > 1$. Then in $\mathbf{N}$

- every unit/vertex $\mathrm{X} \in \mathbf{U}$ has the $\mathrm{outdeg}(\mathrm{X}) \geq 1$ — there is no isolated unit;
- along every walk the values of $w$ are not increasing.

using these two observations we can show that in $\mathbf{N}^*_{NN}$:

- all the values of $w$ on a closed walk are the same and all its arcs are reciprocal — all arcs between units in a nontrivial (at least 2 units) strong component are reciprocal;
- every maximal (can not be extended) elementary (no arc is repeated) walk ends in a RNNs pair;
- there exists at least one RNNs pair – corresponding to $\min_{\mathrm{X},\mathrm{Y} \in \mathbf{U}, \mathrm{X} \neq \mathrm{Y}} d(\mathrm{X}, \mathrm{Y})$.

## Quick agglomerative clustering algorithms

Any graph $\mathbf{G}_{NN}$ is a subgraph of $\mathbf{G}^*_{NN}$. Its connected components are directed (acyclic) trees with a single RNNs pair in the root.

Based on the nearest neighbor graph very efficient $O(n^2)$ algorithms for agglomerative clustering for methods with the reducibility property can be built.

$chain := [\,]$; $\mathbf{W} := \mathbf{U}$;<br>
**while** $\mathrm{card}(\mathbf{W}) > 1$ **do begin**<br>
    **if** $chain = [\,]$ **then** select an arbitrary unit $\mathrm{X} \in \mathbf{W}$ **else** $\mathrm{X} := last(chain)$;<br>
    grow a NN-chain from X until a pair $(\mathrm{Y}, \mathrm{Z})$ of RNNs are obtained;<br>
    agglomerate Y and Z:<br>
        $T := \mathrm{Y} \cup \mathrm{Z}$; $\mathbf{W} := \mathbf{W} \setminus \{\mathrm{Y}, \mathrm{Z}\} \cup \{T\}$; compute $D(T, W), W \in \mathbf{W}$<br>
**end**;

It can be shown that if the clustering method has the reducibility property (minimum, maximum, Ward, ...; but not Bock) then the NN-chain remains a NN-chain also after the agglomeration of the RNNs pair.





# Clustering of Graphs and Networks

When the set of units $\mathbf{U}$ consists of graphs (for example chemical molecules) we speak about *clustering of graphs* (networks). For this purpose we can use standard clustering approaches provided that we have an appropriate definition of dissimilarity between graphs.

The first approach is to define a vector description $[\mathbf{G}] = [g_1, g_2, \ldots, g_m]$ of each graph $\mathbf{G}$, and then use some standard dissimilarity $\delta$ on $\mathbb{R}^m$ to compare these vectors $d(\mathbf{G}_1, \mathbf{G}_2) = \delta([\mathbf{G}_1], [\mathbf{G}_2])$. We can get $[\mathbf{G}]$, for example, by:

*Invariants*: compute the values of selected invariants (indices) on each graph (Trinajstić, 1983).

*Fragments count*: select a collection of subgraphs (fragments), for example triads, and count the number of appearences of each – *fragments spectrum*.





## Invariants and structural properties

Let $\mathbf{Gph}$ be the set of all graphs. An *invariant* of a graph is a mapping $i \colon \mathbf{Gph} \to \mathbb{R}$ which is constant over isomorphic graphs

$$\mathbf{G} \approx \mathbf{H} \Rightarrow i(\mathbf{G}) = i(\mathbf{H})$$

The number of vertices, the number of arcs, the number of edges, maximum degree $\Delta$, chromatic number $\chi$, ... are all graph invariants.

Invariants have an important role in examining the isomorphism of two graphs.

Invariants on *families* of graphs are called *structural properties*: Let $\mathcal{F} \subseteq \mathbf{Gph}$ be a family of graphs. A property $i \colon \mathcal{F} \to \mathbb{R}$ is *structural* on $\mathcal{F}$ iff

$$\forall \mathbf{G}, \mathbf{H} \in \mathcal{F} : (\mathbf{G} \approx \mathbf{H} \Rightarrow i(\mathbf{G}) = i(\mathbf{H}))$$

A collection $\mathcal{I}$ of invariants/structural properties is *complete* iff

$$(\forall i \in \mathcal{I} : i(\mathbf{G}) = i(\mathbf{H})) \Rightarrow \mathbf{G} \approx \mathbf{H}$$

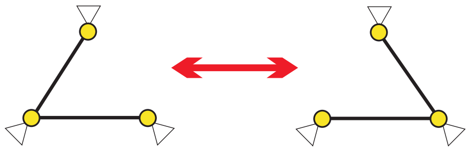
In most cases there is no efficiently computable complete collection.





## Transformations

Different dissimilarities between strings are based on transformations: insert, delete, transpose (Levenshtein 1966, Kashyap 1983). For binary trees Robinson considered a dissimilarity based on the transformation of *neighbors exchange over an edge*.



There is a natural generalization of this approach to graphs and other structured objects (Batagelj 1988): Let $\mathcal{T} = \{T_k\}$ be a set of *basic transformations* of units $T_k : \mathcal{U} \to \mathcal{U}$ and $v : \mathcal{T} \times \mathcal{U} \to \mathbb{R}^+$ value of transformation, which satisfy the conditions:

$$\forall T \in \mathcal{T} : (T : \mathrm{X} \mapsto \mathrm{Y} \Rightarrow \exists S \in \mathcal{T} : (S : \mathrm{Y} \mapsto \mathrm{X} \wedge v(T, \mathrm{X}) = v(S, \mathrm{Y})))$$

and $v(\mathrm{id}, X) = 0$.

## Transformations based dissimilarity

Suppose that for each pair $\mathrm{X}, \mathrm{Y} \in \mathcal{U}$ there exists a finite sequence $\tau = (T_1, T_2, \ldots, T_t)$ such that: $\tau(\mathrm{X}) = T_t \circ T_{t-1} \circ \ldots \circ T_1(X) = \mathrm{Y}$. Then we can define:

$$d(\mathrm{X}, \mathrm{Y}) = \min_{\tau}(v(\tau(\mathrm{X})) : \tau(\mathrm{X}) = \mathrm{Y})$$

where

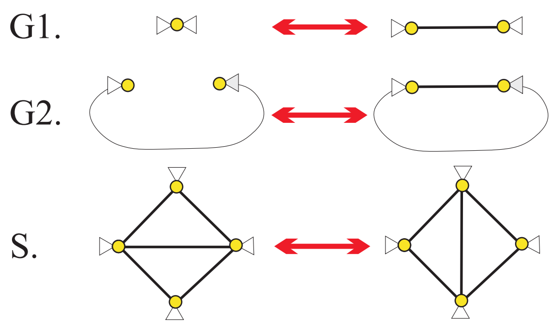
$$v(\tau(X)) = \begin{cases} 0 & \tau = \mathrm{id} \\ v(\eta(T(\mathrm{X}))) + v(T, \mathrm{X}) & \tau = \eta \circ T \end{cases}$$

It is easy to verify that so defined dissimilarity $d(\mathrm{X}, \mathrm{Y})$ is a distance.





## Examples of transformations




Using the transformations G1 and G2 we can transform any pair of connected simple graphs one to the other. For triangulations of the plane on $n$ vertices S is such a transformation.





# Clustering in Graphs and Networks

Since in a graph $\mathbf{G} = (V, L)$ we have two kinds of objects – vertices and links we can speak about *clustering of vertices* and *clustering of links*. Usually we deal with clustering of vertices.

Again we can use the standard clustering methods provided that we have an appropriate definition of dissimilarity between vertices.

The usual approach is to define a vector description $[v] = [t_1, t_2, \ldots, t_m]$ of each vertex $v \in V$, and then use some standard dissimilarity $\delta$ on $\mathbb{R}^m$ to compare these vectors $d(u, v) = \delta([u], [v])$. For some 'nonstandard' such descriptions see Moody (2001) and Harel, Koren (2001).

We can assign to each vertex $v$ also different neighborhoods

$$N(v) = \{u \in V : (v, u) \in L\}$$

and other sets. In these cases the dissimilarities between sets are used on them.





## Properties of vertices

For a given graph $\mathbf{G} = (V, L)$ a property $t : V \to \mathbb{R}$ is *structural* iff for every automorphism $\varphi$ of $\mathbf{G}$ it holds

$$\forall v \in V : t(v) = t(\varphi(v))$$

Examples of such properties are

$t(v) =$ degree (number of neighbors) of vertex $v$
$t(v) =$ number of vertices at distance $d$ from vertex $v$
$t(v) =$ number of triads of type $x$ at vertex $v$

## Properties of pairs of vertices

For a given graph $\mathbf{G} = (V, L)$ a *property of pairs of vertices* $q : V \times V \to \mathbb{R}$ is *structural* if for every automorphism $\varphi$ of $\mathbf{G}$ it holds

$$\forall u, v \in V : q(u, v) = q(\varphi(u), \varphi(v))$$

Some examples of structural properties of pairs of vertices

$q(u, v) =$ **if** $(u, v) \in L$ **then** 1 **else** 0
$q(u, v) =$ number of common neighbors of units $u$ and $v$
$q(u, v) =$ length of the shortest path from $u$ to $v$

Using a selected property of pairs of vertices $q$ we can describe each vertex $u$ with a vector

$$[u] = [q(u, v_1), q(u, v_2), \dots, q(u, v_n), q(v_1, u), \dots, q(v_n, u)]$$

and again define the dissimilarity between vertices $u, v \in V$ as $d(u, v) = \delta([u], [v])$.





## Matrix dissimilarities

The following is a list of dissimilarities, used in literature, based on properties of pairs of vertices for measuring the similarity between vertices $v_i$ and $v_j$ $(p \geq 0)$:

Manhattan: $d_m(v_i, v_j) = \sum_{s=1}^{n} (|q_{is} - q_{js}| + |q_{si} - q_{sj}|)$

Euclidean: $d_E(v_i, v_j) = \sqrt{\sum_{s=1}^{n} ((q_{is} - q_{js})^2 + (q_{si} - q_{sj})^2)}$

Truncated Man.: $d_s(v_i, v_j) = \sum_{\substack{s=1 \\ s \neq i,j}}^{n} (|q_{is} - q_{js}| + |q_{si} - q_{sj}|)$

Truncated Euc.: $d_S(v_i, v_j) = \sqrt{\sum_{\substack{s=1 \\ s \neq i,j}}^{n} ((q_{is} - q_{js})^2 + (q_{si} - q_{sj})^2)}$

Corrected Man.: $d_c(p)(v_i, v_j) = d_s(v_i, v_j) + p \cdot (|q_{ii} - q_{jj}| + |q_{ij} - q_{ji}|)$

Corrected Euc.: $d_e(p)(v_i, v_j) = \sqrt{d_S(v_i, v_j)^2 + p \cdot ((q_{ii} - q_{jj})^2 + (q_{ij} - q_{ji})^2)}$

Corrected diss.: $d_C(p)(v_i, v_j) = \sqrt{d_c(p)(v_i, v_j)}$

The corrected dissimilarities with $p = 1$ should be used.





# Graph theory approaches

The basic decomposition of graphs is to (weakly) connected components – partition of vertices (and links); and to (weakly) biconnected components – partition of links. For both very efficient algorithms exist.

From a network $\mathbf{N} = (V, L, w)$ we can get for a treshold $t$ a layer network $\mathbf{N}(t) = (V, L_t, w)$ where $L_t = \{p \in L : w(p) \geq t\}$. From it we can get a clustering $\mathbf{C}(t)$ with connected components as clusters. For different tresholds these clusterings form a hierarchy.

In seventies and eighties Matula studied different types of connectivities in graphs and structures they induce. In most cases the algorithms are too demanding to be used on larger graphs. A recent overview of connectivity algorithms was made by Esfahanian.

For directed graphs the fundamental decomposition results can be found in Harary, Norman, and Cartwright (1965).





## Decomposition of directed graphs

Given a simple directed graph $\mathbf{G} = (V, R)$, $R \subseteq V \times V$ we introduce two new relations, $R^\star$ (*transitive and reflexive closure*) and $\overline{R}$ (*transitive closure*), based on $R$:

$$uR^\star v := \exists k \in \mathbb{N} : uR^k v \qquad \text{and} \qquad u\overline{R}v := \exists k \in \mathbb{N}^+ : uR^k v$$

or equivalently

$$R^\star = \bigcup_{k \in \mathbb{N}} R^k \qquad \text{and} \qquad \overline{R} = \bigcup_{k \in \mathbb{N}^+} R^k$$

**Theorem 3.1**
*a)* $uR^k v$ *iff in the graph* $\mathbf{G} = (V, R)$ *there exists a walk of length* $k$ *from* $u$ *to* $v$.
*b)* $uR^\star v$ *iff in the graph* $\mathbf{G} = (V, R)$ *there exists a walk from* $u$ *to* $v$.
*c)* $u\overline{R}v$ *iff in the graph* $\mathbf{G} = (V, R)$ *there exists a non-null walk from* $u$ *to* $v$.

## Acyclic Relations

A relation $R \in V \times V$ is *acyclic* if and only if

$$\forall v \in V \, \forall k > 0 : \neg v(R \setminus I)^k v$$

i.e. if its graph, except for loops, contains no cycles. This condition can be written also in the form $\overline{(R \setminus I)} \cap I = \emptyset$. We shall denote by $\mathrm{Acy}(V)$ the set of all acyclic relations on $V$.

A relation $R \in V \times V$ is *strictly acyclic* if and only if

$$\forall v \in V \, \forall k > 0 : \neg v R^k v$$

i.e. if its graph contains no cycles and, also, loops are not allowed. This condition can be written also in form $\overline{R} \cap I = \emptyset$. Each strictly acyclic relation is also acyclic.

**Theorem 3.2** *For an acyclic relation $R \in \mathrm{Acy}(V)$ over a finite, nonempty set $V$ there is at least one minimal, $R^{-1}(v) \subseteq \{v\}$, and at least one maximal, $R(v) \subseteq \{v\}$, element.*





## Factorization

Suppose that on the set $V$ we have a relation $R \in V \times V$ and an equivalence $\sim$. The equivalence $\sim$ partitions the set $V$ into equivalence classes which form the family $V/\!\sim$. In $V/\!\sim$ we can define the *factor* relation $\mathbf{r} = R/\!\sim$

$$\mathbf{r} = R/\!\sim := \{\exists x \in X \exists y \in Y : xRy\}$$

We will see, later, that all blockmodels can be described in these terms. The factor relation is the image of a blockmodel.

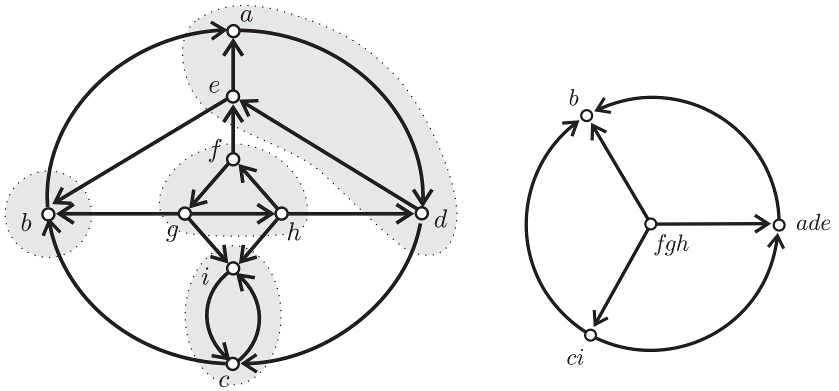
For a relation $R \in V \times V$ the strong connectivity relation $R^S = R^\star \cap (R^{-1})^\star$ is an equivalence. It partitions the set $V$ into equivalence classes (strong components) which form a family $V/R^S$.

**Theorem 3.3** *Let $R \in V \times V$. The relation $\sqsubseteq := R/R^S$ is acyclic on $V/R^S$.*
*If $R$ is a preorder (transitive and reflexive) then $\sqsubseteq$ is a partial order on $V/R^S$.*
*If $R$ is a tournament (asymmetric and comparable) then $\sqsubseteq$ is a linear order on $V/R^S$.*





## Graph, strong components and factorization







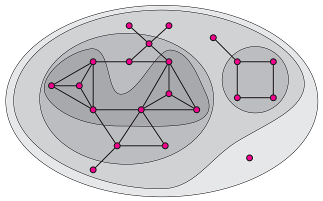

## Cores

The notion of a core was introduced by Seidman in 1983.

In a given graph $\mathbf{G} = (V, L)$ a subgraph $\mathbf{H}_k = (W, L|W)$ induced by the set $W$ is a $k$-*core* or a *core of order* $k$ iff $\forall v \in W : \deg_H(v) \geq k$, and $\mathbf{H}_k$ is the maximum subgraph with this property. The core of maximum order is also called the *main* core. The *core number* of vertex $v$ is the highest order of a core that contains this vertex.

The degree $\deg(v)$ can be: in-degree, out-degree, in-degree + out-degree, ...determining different types of cores.

In figure an example of cores decomposition of a given graph is presented. We can see the following properties of cores:



- The cores are nested: $i < j \implies \mathbf{H}_j \subseteq \mathbf{H}_i$
- Cores are not necessarily connected subgraphs.

## Determining and using cores

A very efficient $O(m)$ algorithm (Batagelj, Zaveršnik 2002) for determining the cores hierarchy can be built based on the following property:

> If from a given graph $\mathbf{G} = (V, L)$ we recursively delete all vertices, and lines incident with them, of degree less than $k$, the remaining graph is the $k$-core.

The notion of cores can be generalized to networks.

Using cores we can identify the densiest parts of a graph. For revealing the internal structure of the main core we can use standard clustering procedures on dissimilarities between vertices. Afterwards we can remove the links of the main core and analyse the residium graph.

Cores can be used also to localize the search for some computationally more demanding substructures.





## Short cycles

A subgraph $\mathbf{H} = (V', A')$ of $\mathbf{G} = (V, A)$ is *cyclic $k$-gonal* if each its vertex and each its edge belong to at least one cycle of length at most $k$ and at least 2 in $\mathbf{H}$.

A sequence $(C_1, C_2, \ldots, C_s)$ of cycles of length at most $k$ (and at least 2) of $\mathbf{G}$ *cyclic $k$-gonally connects* vertex $u \in V$ to vertex $v \in V$ iff $u \in C_1$ and $v \in C_s$ or $u \in C_s$ and $v \in C_1$ and $V(C_{i-1}) \cap V(C_i) \neq \emptyset$, $i = 2, \ldots s$; such sequence is called a *cyclic $k$-gonal chain*.

A pair of vertices $u, v \in V$ is *cyclic $k$-gonally connected* iff $u = v$, or there exists a cyclic $k$-gonal chain that connects $u$ to $v$.

**Theorem 3.4** *Cyclic $k$-gonal connectivity is an equivalence relation on the set of vertices $V$.*

An arc is *cyclic* iff it belongs to some cycle (of any length) in the graph $\mathbf{G}$.

**Theorem 3.5** *If in the graph $\mathbf{G}$ for each cyclic arc the length of a shortest cycle that contains it is at most $k$ then the cyclic $k$-gonal reduction of $\mathbf{G}$ is an acyclic graph.*





## Final remarks

The agglomerative methods can be adapted for large sparse networks in sense of relational constraint clustering – we have to compute dissimilarities only between units/vertices connected by a link.

The Sollin's MST algorithm can be very efficiently implemented for large sparse networks.